International Requirements Engineering Board

Thorsten Cziharz
Peter Hruschka
Stefan Queins
Thorsten Weyer

Handbuch der Anforderungsmodellierung nach IREB Standard

Aus- und Weiterbildung zum
IREB Certified Professional for Requirements Engineering
Advanced Level "Requirements Modeling"

Version 1.3 31. August 2016

Die Erarbeitung dieses Handbuches wurde unterstützt durch:

PALUNO The Ruhr Institute for Software Technology

SOPHIST

Nutzungsbedingungen

1. Jede Einzelperson und jeder Seminaranbieter darf dieses Handbuch als Grundlage für Seminare verwenden, sofern die Inhaber der Urheberrechte als Quelle und Besitzer des Urheberrechts anerkannt und benannt werden. Des Weiteren darf dieses Handbuch zu Werbungszwecken nur mit Einwilligung des IREB e.V. verwendet werden.

2. Jede Einzelperson oder Gruppe von Einzelpersonen darf dieses Handbuch als Grundlage für Artikel, Bücher oder andere abgeleitete Veröffentlichungen verwenden, sofern die Autoren und das IREB e.V. als Quelle und Besitzer des Urheberrechts genannt werden.

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Die Verwertung ist – soweit sie nicht ausdrücklich durch das Urheberrechtsgesetz (UrhG) gestattet ist – nur mit Zustimmung der Berechtigten zulässig. Dies gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmung, Einspeicherung und Verarbeitung in elektronischen Systemen und öffentliche Zugänglichmachung.

Danksagung

Wir danken Torsten Bandyszak, Nelufar Ulfat-Bunyadi und Stefan Sturm für die Unterstützung bei der Erstellung dieses Manuskripts.

Dieses Handbuch wurde erstellt von (in alphabetischer Reihenfolge):
Thorsten Cziharz, Dr. Peter Hruschka, Dr. Stefan Queins und Dr. Thorsten Weyer

Urheberrecht © 2016 des Handbuchs der Anforderungsmodellierung nach IREB Standard besitzen die aufgeführten Autoren. Die Rechte sind übertragen auf das IREB International Requirements Engineering Board e.V.

Inhaltsverzeichnis

1 Grundlagen ... 1
1.1 Warum Anforderungen modellieren? ... 1
1.2 Ausprägungen der Anforderungsmodellierung ... 2
1.3 Begriffe und Konzepte in der Modellierung von Anforderungen ... 2
1.4 Anforderungsmodelle ... 4
1.5 Sichten der Anforderungsmodellierung ... 7
1.6 Sichten der dynamischen Sicht in der Anforderungsmodellierung ... 9
1.7 Anpassung von Modellierungssprachen zur Anforderungsmodellierung ... 10
1.8 Integration textueller Anforderungen in das Anforderungsmodell ... 11
1.9 Dokumentation von Abhängigkeiten zwischen Modellelementen ... 11
1.10 Vorteile der Anforderungsmodellierung ... 12
1.11 Qualität von Anforderungsmodellen ... 13
1.12 Vertiefende Literatur ... 15
2 Kontextmodellierung ... 17
2.1 Zweck ... 17
2.2 Kontextdiagramme ... 17
2.3 Andere Arten der Kontextmodellierung ... 20
2.4 Vertiefende Literatur ... 20
3 Informationsstrukturmodellierung ... 21
3.1 Zweck ... 21
3.2 Modellierung von Informationsstrukturen ... 21
3.3 Einfaches Beispiel ... 22
3.4 Modellierung von Klassen, Attributen und Datentypen ... 23
3.5 Modellierung von Beziehungen ... 32
3.6 Modellierung von Generalisierungen und Spezialisierungen ... 39
3.7 Weiterführende Modellierungskonzepte ... 42
3.8 Vertiefende Literatur ... 42
4 Dynamische Sichten ... 45
4.1 Dynamische Sichten der Anforderungsmodellierung ... 45
4.2 Use-Case-Modellierung ... 45
4.3 Datenfluss- und kontrollflussorientierte Modellierung von Anforderungen ... 53
4.4 Zustandsorientierte Modellierung von Anforderungen ... 66
4.5 Vertiefende Literatur ... 84
5 Szenariomodellierung ... 87
5.1 Zweck ... 87
5.2 Zusammenhang zwischen Szenarien und Use-Cases ... 88
5.3 Ansätze zur Szenariomodellierung ... 89
5.4 Einfache Beispiele für ein modelliertes Szenario ... 89
5.5 Szenariomodellierung mit Sequenzdiagrammen ... 91
5.6 Szenariomodellierung mit Kommunikationsdiagrammen ... 100
5.7 Beispiele für typische Diagramme in der Szenariosicht ... 100
5.8 Vertiefende Literatur ... 104
Glossar ... 105
Abkürzungsverzeichnis ... 111
Literaturverzeichnis ... 113
Index ... 117

Das IREB CPRE Advanced Level Modul "Requirements Modeling"

In den vergangenen Jahren sind der Umfang und die Komplexität typischer softwarebasierter Systeme erheblich angestiegen, was sich unmittelbar auf die Anzahl von Anforderungen und auf die Komplexität hinsichtlich der wechselseitigen Abhängigkeiten zwischen Anforderungen niederschlägt. Sämtliche Prognosen über den zukünftig zu erwartenden Anstieg von Umfang und Komplexität softwarebasierter Systeme lassen erwarten, dass die Menge der Anforderungen und die Komplexität der wechselseitigen Abhängigkeiten in Zukunft nochmals in dramatischer Weise zunehmen werden. Dies wird z.B. deutlich, wenn man die Entwicklungstrends im Bereich betrieblicher Informationssysteme hinsichtlich des Internet-der-Dienste (IoS) und Internet-der-Dinge (IoT) oder die Entwicklung auf dem Gebiet intelligenter eingebetteter Systeme betrachtet. Beide Trends sind Wegbereiter für eine in gewisser Weise revolutionäre Durchdringung der physikalischen Welt mit dynamisch vernetzten softwarebasierten Systemen, den so genannten „Cyber-Physical Systems".

Aufgrund der Tatsache, dass zum einen Anforderungen eine zentrale Rolle im Entwicklungsprozess softwarebasierter Systeme einnehmen und zum anderen der Umfang und die Komplexität der Anforderungen eines Systems immer schwieriger zu handhaben ist, stößt die Spezifikation von Anforderungen durch ausschließlich natürlichsprachliche (d.h. textuelle) Aussagen in vielen Bereichen bereits heute an ihre Grenzen, was sich in vielen Fällen nachhaltig negativ auf die entsprechenden Entwicklungsprojekte auswirkt. Durch die vielfältigen Vorteile der Nutzung grafischer Modelle hinsichtlich Lesbarkeit, Komplexitätsbeherrschung, automatischer Analysierbarkeit und Weiterverarbeitung umfangreicher und komplexer Sachverhalte, nimmt die grafische Modellierung von Anforderungen immer stärker Einzug in die Praxis.

Das Advanced-Level-Modul „Requirements Modeling" des IREB Certified Professional for Requirements Engineering vermittelt das Rüstzeug, um Anforderungen umfangreicher und komplexer Systeme durch den Einsatz standardisierter und weit verbreiteter Modellierungssprachen spezifizieren zu können. Für diese Modellierungssprachen existiert eine umfassende Werkzeugunterstützung, beginnend bei Freeware-Werkzeugen bis hin zu leistungsfähigen kommerziellen Werzeugen, die ein weitrechendes Automatisierungspotenzial und eine bruchfreie Integration mit anderen in Entwicklungsprozessen eingesetzten Werkzeugen (z.B. zum Projekt- und Testmanagement) bieten.

Weitere Informationen zum IREB Certified Professional for Requirements Engineering – Advanced Level „Requirements Modeling" finden sich auf: http://www.ireb.org

Vorwort

Das vorliegende *Handbuch der Anforderungsmodellierung nach IREB Standard* ergänzt den Lehrplan des International Requirements Engineering Boards für das Advanced-Level-Modul „Requirements Modeling“ Version 2.0.

Das Handbuch richtet sich sowohl an Trainingsanbieter, die Seminare zur Anforderungsmodellierung nach dem IREB-Standard anbieten wollen, als auch an Trainingsteilnehmer und interessierte Praktiker, die einen detaillierten Einblick in den Lehrstoff dieses Advanced-Level-Moduls und in die Anforderungsmodellierung nach IREB Standard erhalten möchten.

Dieses Handbuch ist **kein** Ersatz für Schulungen zu diesem Thema. Es stellt vielmehr ein Bindeglied zwischen dem komprimiert gehaltenen Lehrplan (der lediglich die Lernziele des Moduls nennt und erläutert) und der Vielzahl an Literatur dar, die zum Thema Anforderungsmodellierung in den letzten Jahrzehnten entstanden ist.

Die in diesem Handbuch beschriebenen Inhalte, zusammen mit den Hinweisen auf vertiefende Literatur, sollen Trainingsanbieter dabei unterstützen, die Trainingsteilnehmer fokussiert auf die Zertifizierungsprüfung vorzubereiten. Trainingsteilnehmern und interessierten Praktikern bietet dieses Handbuch die Möglichkeit, die Kenntnisse im Bereich der Anforderugnsmodellierung zu vertiefen und die detaillierten Inhalte auf Basis der Literaturempfehlungen aufzuarbeiten. Darüber hinaus soll dieses Handbuch auch als Referenz dienen, um z.B. nach erfolgreicher Zertifizierung das Wissen über die verschiedenen Themenbereiche der Anforderungsmodellierung auffrischen zu können.

Verbesserungs- und Korrekturvorschläge sind jederzeit herzlich willkommen!

E-Mail-Kontakt: requirementsmodeling.guide@ireb.org

Wir wünschen Ihnen viel Spaß beim Studium dieses Handuchs und viel Erfolg bei der Zertifizierungsprüfung zum IREB Certified Professional for Requirements Engineering - Advanced Level „Requirements Modeling“.

Thorsten Cziharz (Sophist GmbH)
Peter Hruschka (Atlantic Systems Guild)
Stefan Queins (Sophist GmbH)
Thorsten Weyer (paluno - The Ruhr Institute for Software Technology)

Sommer 2016

Versions Historie

Version	Datum	Kommentar
1.0	Oktober 2013	Release Version 1.0
1.0-1	11. November 2013	Kapitel 1.4.1: Anzahl der Diagrammtypen korrigiert Syntax in Literaturverweis HARE korrigiert Versions Historie eingeführt
1.0-2	29. November 2013	Gemeldete Typos korrigiert
1.0-3	1. März 2014	Inhaltliches Feedback eingearbeitet
1.0-4	23. Juli 2014	Typo korrigiert
1.1	29. Februar 2014	Kleinere inhaltliche Anpassungen vorgenommen
1.2	17. Juli 2016	Kleinere Anpassung hinsichlich des Englischen Handbuchs vorgenommen.
1.3	31. August 2016	Inhalte zur Modellierung von Assoziationsklassen augenommen und kleiner Anpassungen vorgenommen.

1 Grundlagen

Anforderungen nehmen im Lebenszyklus von Systemen eine zentrale Rolle ein. Insbesondere die verschiedenen Entwicklungsdisziplinen (wie z.B. Architekturentwurf, Design, Implementierung und Test) stützen sich elementar auf die im Requirements Engineering spezifizierten Anforderungen des Systems und sind in erheblichem Umfang von der Qualität dieser Anforderungen abhängig. Neben den Entwicklungsdisziplinen hängen aber auch Aktivitäten in der Wartung und Pflege bis hin zur Außerbetriebnahme des Systems sowie der Entwicklung vorgelagerten Aktivitäten, wie z.B. die Abschätzung von Risiken und Aufwand des Entwicklungsprojektes, wesentlich von den Anforderungen und deren Qualität ab.

Nach dem *IREB Glossary of Requirements Engineering Terminology* [Glin2011] ist eine Anforderung (1) ein Bedürfnis, welches von einem Stakeholder wahrgenommen wird oder (2) eine Fähigkeit oder Eigenschaft, die ein System aufweisen muss. Das Requirements Engineering ist dafür verantwortlich, dass die Anforderungen des zu entwickelnden Systems möglichst vollständig, korrekt und präzise formuliert sind, um dadurch die anderen Entwicklungsdisziplinen und Aktivitäten im Lebenszyklus des Systems möglichst optimal zu unterstützen.

1.1 Warum Anforderungen modellieren?

Abbildung 1 zeigt an einem stark vereinfachten Beispiel den Unterschied zwischen textuellen und modellierten Anforderungen. Die Abbildung zeigt im linken Bereich vier textuelle Anforderungen, die ein notwendiges Verhalten in Bezug auf die Eingabe von Daten über eine Eingabemaske spezifizieren. Der rechte Bereich zeigt ein Anforderungsdiagramm, in dem die entsprechenden Anforderungen modelliert sind.

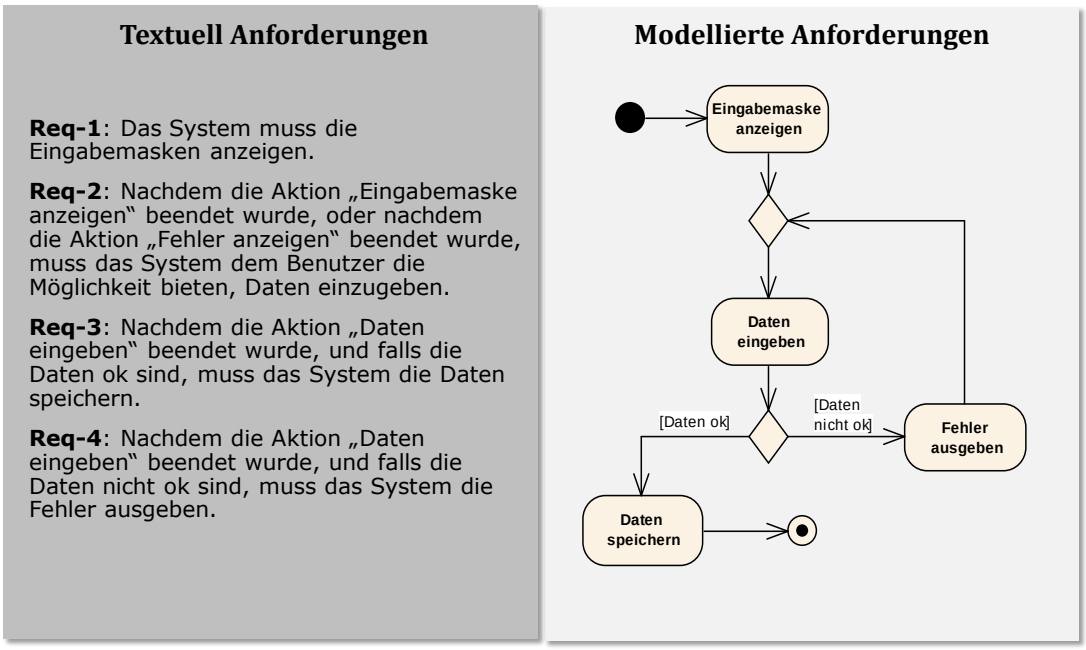

Abbildung 1: Textuelle Anforderungen vs. modellierte Anforderungen

Wie bereits an diesem einfachen Beispiel zu erkennen ist, wird durch die modellierten Anforderungen das notwendige Verhalten des Systems strukturierter und verständlicher dar-

gestellt und kann in einfacher Weise Stück für Stück vom Leser nachvollzogen werden. In dem einfachen Beispiel wird darüber hinaus deutlich, dass auch der Gesamtzusammenhang der verschiedenen Aspekte des notwendigen Systemverhaltens in den modellierten Anforderungen explizit ersichtlich ist, wohingegen solche Informationen in den textuellen Anforderungen nur implizit vorhanden sind (vgl. auch [Davi1993]). Typischerweise besitzen Softwaresysteme heutzutage bedeutend komplexere Abläufe, wodurch die zugehörigen textuellen Anforderungen sehr umfangreich und komplex werden. Die Zusammenhänge innerhalb solcher komplexer Abläufe sind dann auf Basis der textuellen Anforderungen vom Leser schwer nachzuvollziehen.

1.2 Ausprägungen der Anforderungsmodellierung

Die Modellierung von Anforderungen kommt heute im Requirements Engineering auf unterschiedliche Weisen zum Einsatz:

1.2.1.1 Modellierung von Anforderungen als Mittel zur Spezifikation von Anforderungen

In diesem Falle treten Anforderungsdiagramme an die Stelle textuell spezifizierter Anforderungen, d.h. Anforderungsdiagramme werden als primäres Mittel verwendet, um die Anforderungen des Systems oder Teile der Anforderungen des Systems zu spezifizieren. Die Anforderungsdiagramme können (und sollten) um textuelle Anforderungen oder textuelle Erläuterungen ergänzt werden, und zwar dann, wenn ein Text kompakter oder einfacher zu handhaben ist als Diagramme. Werden trotzdem alle Anforderungen auch in textueller Form benötigt (z.B. weil Vertragsbedingungen oder Zertifizierungsvorgaben dies vorsehen), können diese aus den Anforderungsdiagrammen generiert werden, indem z.B. Schablonen für die Überführung von Anforderungsdiagrammen in textuell spezifizierten Anforderungen verwendet werden.

1.2.1.2 Modellierung existierender textueller Anforderungen zum Zwecke der Prüfung

In diesem Falle wird für eine logisch zusammenhängende Menge textuell spezifizierter Anforderungen, die z.B. ein notwendiges komplexes Verhalten des Systems spezifizieren, ein Anforderungsdiagramm erstellt, um z.B. die Verständlichkeit der textuellen Anforderungen zu prüfen oder um Widersprüche oder Unvollständigkeiten in den textuellen Anforderungen aufzudecken. Aufgedeckte Defekte werden dann in den textuellen Anforderungen korrigiert.

1.2.1.3 Modellierung existierender textueller Anforderungen zur besseren Verständlichkeit

In diesem Falle werden z.B. durch modellierte Anforderungen umfangreiche und komplexe Zusammenhänge, die das Verhalten des Systems betreffen, im Zusammenhang dargestellt. Diese redundante Form der Spezifikation kann jedoch zu erheblichen Problemen im Hinblick auf Widersprüche zwischen textuell spezifizierten und modellierten Anforderungen führen.

1.3 Begriffe und Konzepte in der Modellierung von Anforderungen

Im Folgenden werden auf der Grundlage allgemeiner Begriffe und Konzepte in der Systemmodellierung die in der Modellierung von Anforderungen relevanten Begriffe und Konzepte sowie deren wesentliche Beziehungen zueinander betrachtet. Abbildung 2 zeigt ein semantisches Netz der für die Anforderungsmodellierung relevanten grundlegenden Begriffe und

Konzepte. Begriffe, die bereits im *IREB Glossary of Requirements Engineering Terminology* definiert sind, wurden in der Abbildung mit ↑ gekennzeichnet.

Das Begriffssystem baut auf verschiedenen Definitionen im *IREB Glossary of Requirements Engineering Terminology* [Glin2011] auf und ergänzt dieses um Begriffe und Konzepte, die speziell für die Anforderungsmodellierung wesentlich sind. Ein *Modell* wird als ein abstrahierendes Abbild von Eigenschaften eines *Systems* verstanden. Um Umfang und Komplexität der Modellbildung handhabbar zu machen, werden verschiedene *Sichten* (Views) auf das *System* (und dessen Umgebung) eingenommen und die Eigenschaften des Systems in Bezug auf die jeweils spezifische Sicht durch *Diagramme* und ergänzende *textuelle* Modellelemente repräsentiert. Jedem Diagramm liegt jeweils ein spezifischer *Diagrammtyp* zugrunde, der wiederum über eine *Modellierungssprache* (genauer über *Syntax*, *Semantik und ggf. Pragmatik*) definiert ist. Die einem *Diagrammtyp* zugrunde liegende *Modellierungssprache* definiert die Menge von *Modellierungskonstrukten*, die zur Konstruktion der entsprechenden *Diagramme* verwendet werden können (z.B. Klasse und Assoziation zur Konstruktion von Klassendiagrammen). In einer Modellierungssprache sind für die Modellierungskonstrukte grafische und/oder textuelle Notationen definiert.

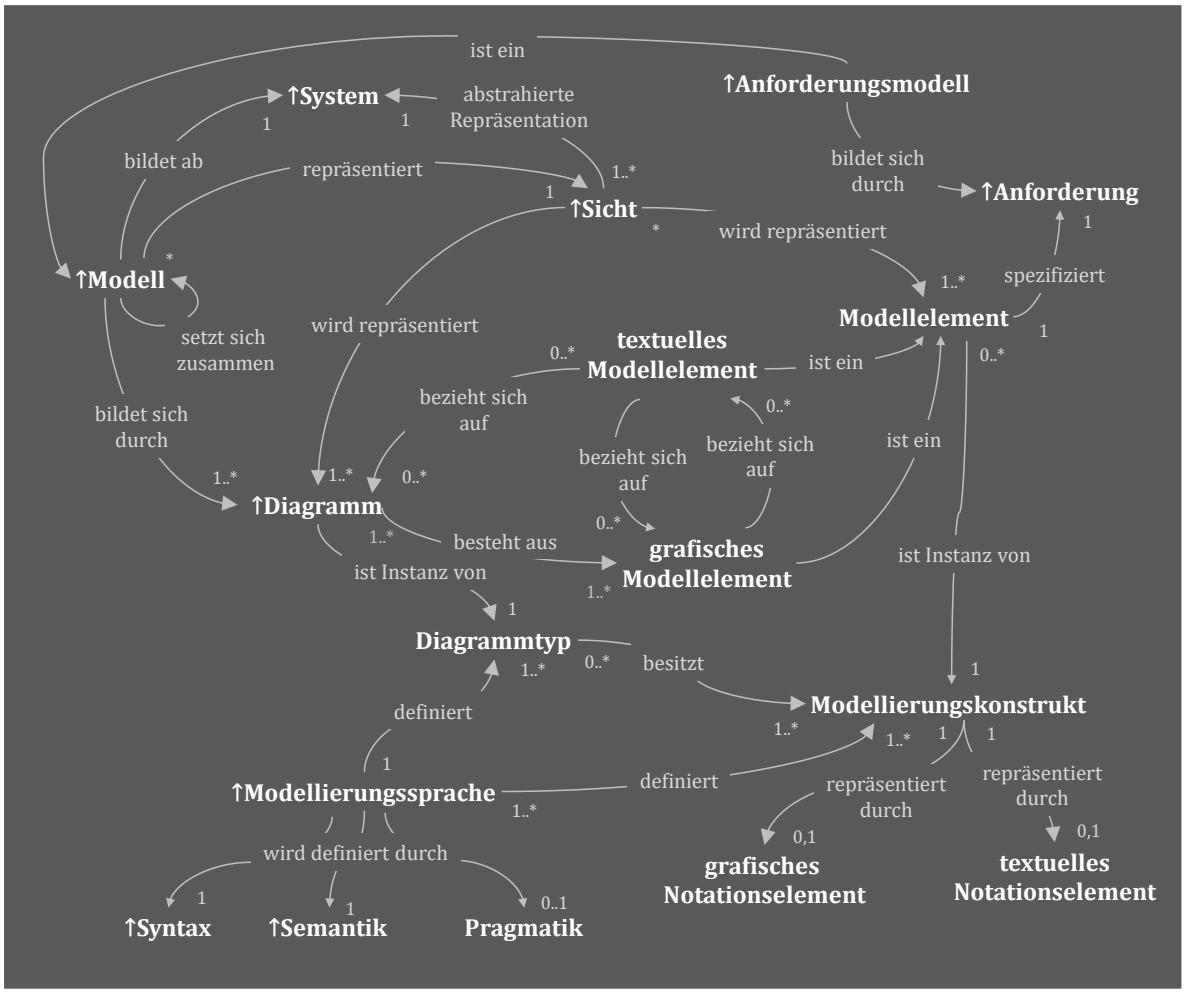

Abbildung 2: Begriffsnetz der Terminologie in der Anforderungsmodellierung

Ein *Diagramm* setzt sich aus einer Menge von *Modellelementen* zusammen, die jeweils die Ausprägung eines spezifischen *grafischen Modellierungskonstrukts* der *Modellierungssprache* des zugehörigen *Diagrammtyps* sind (z.B. Klasse „Mitarbeiter“, Assoziation „ist beschäftigt bei“, Klasse „Unternehmen“). *Diagramme* und grafische Modellelemente können durch *tex-*

tuelle Modellelemente (z.B. textuelle Beschreibung des Auslösers eines Use-Cases) ergänzt werden, die wiederum Ausprägungen spezifischer *textueller Modellierungskonstrukte* (z.B. Abschnitt einer Use-Case-Schablone) sind. Die grafischen und textuellen *Modellelemente* bilden die atomaren Bestandteile von *Modellen*. Ein *Anforderungsmodell* ist eine spezifische Art von Modellen (genauer: von Systemmodellen), durch die die *Anforderungen* eines Systems mit Hilfe von *Diagrammen* und textueller Ergänzung spezifiziert werden.

1.4 Anforderungsmodelle

Die einzelnen Anforderungen des Anforderungsmodells werden dabei durch Modellelemente repräsentiert, die innerhalb von Anforderungsdiagrammen sowie durch textuelle Ergänzungen dieser Diagramme spezifiziert sind.

1.4.1 Modellierungssprachen für die Anforderungsmodellierung

Zur Anforderungsmodellierung existieren eine Reihe von Diagrammtypen und zugehörige Modellierungssprachen. Die Auswahlentscheidung für die jeweils verwendeten Diagrammtypen hängt dabei vom jeweiligen Zweck ab, der sich dadurch bestimmt, welche spezifischen Anforderungen des Systems dokumentiert werden sollen und welche Personen die „Adressaten“ der Anforderungsmodelle sind.

Die Relevanz verschiedener Diagrammtypen hängt dabei oftmals auch von der Art des Systems ab (z.B. betriebliches Informationssystem oder eingebettetes System) und teilweise auch von der Anwendungsdomäne (z.B. Banken, Versicherungen, Automatisierungstechnik, Fahrzeug-, Flugzeugbau) ab, für die das System entwickelt wird. So bildet beispielsweise bei eingebetteten Systemen häufig das reaktive Verhalten des Systems den Schwerpunkt der Anforderungsmodellierung. Die Ursache hierfür liegt darin, dass Umfang und Komplexität des notwendigen Verhaltens eines heutigen eingebetteten Systems hauptsächlich durch die notwendige Reaktivität des Systems bestimmt wird. Deshalb werden zur Anforderungsmodellierung in der Entwicklung eingebetteter Systeme Zustandsmaschinendiagramme der OMG SysML [OMG2010a], OMG UML [OMG2010b] oder auch MATLAB/Simulink Stateflow-Diagramme (z.B. [Hoff1998]) eingesetzt, die dann durch komplementäre Diagramme ergänzt werden, wie z.B. Use-Case-Diagramme, Szenarien oder Aktivitätsdiagrammen. Demgegenüber besitzen betriebliche Informationssysteme (z.B. Software zur Bearbeitung von Kreditanfragen) in der Regel kein umfangreiches und komplexes reaktives Verhalten. Der Umfang und die Komplexität des notwendigen Systemverhaltens werden hier durch die teilweise sehr komplexe Ablauflogik (Kontrollfluss) von Aktivitäten und durch die oftmals sehr umfangreichen und komplexen Informationsstrukturen bestimmt. Aus diesem Grund finden sich in der Anforderungsmodellierung solcher Systeme gegenwärtig zum großen Teil Diagrammtypen, mit denen umfangreiche und komplexe Informationsstrukturen modelliert werden können (z.B. UML Klassendiagramme) sowie Diagrammtypen, durch die ablauforientierte Aspekte modelliert werden können (wie z.B. Ereignisgesteuerte Prozessketten [Sche2001]) oder BPMN-Diagramme [OMG2011] im Rahmen der Business-Analyse bzw. Aktivitätsdiagramme der UML, um z.B. Anforderungen in Bezug auf die notwendige Ablauflogik des zu entwickelnden Softwaresystems zu modellieren. Auch hier können wiederum ergänzend andere Diagrammtypen eingesetzt werden, wie z.B. Zustandsmaschinendiagramme, um Anforderungen im Hinblick auf die notwendige Reaktivität des Systems zu modellieren.

Neben spezifischen Ansätzen wie Ereignisgesteuerte Prozessketten (EPKs) oder BPMN, die oft im Rahmen der Business-Analyse verwendet werden oder MATLAB/Simulink-Diagrammen in der Anforderungsmodellierung für eingebettete Systeme, werden sehr häufig die

weit verbreiteten „universellen“ Modellierungsansätze UML und SysML zur Modellierung von Anforderungen eingesetzt. Die UML in der Version 2.4 unterscheidet zwischen 14 verschiedene Diagrammtypen, von denen sieben Diagrammtypen der Strukturmodellierung und sieben Diagrammtypen der Verhaltensmodellierung dienen, wobei der Diagrammtyp „Profildiagramm“ zur Dokumentation von Sprachprofilen (d.h. Adaptionen, Erweiterungen der Modellierungssprache) und nicht, wie die übrigen Diagrammtypen, zur Systemmodellierung im eigentlichen Sinne verwendet wird. Die SysML wurde speziell für die Modellierung in der Entwicklung komplexer Systeme konzipiert und ist eine um spezielle Diagrammtypen und Notationselemente erweiterte Untermenge der UML. Die entsprechenden Erweiterungen beziehen sich dabei auf neue Strukturdiagramme (Interne Blockdiagramme, Blockdefinitionsdiagramme, Zusicherungsdiagramme). Die SysML besitzt keinen Diagrammtyp „Klassendiagramm“ mehr. Hinsichtlich der Verhaltensdiagramme werden in der SysML keine neuen Diagrammtypen eingeführt, sondern die Verhaltensdiagrammtypen der UML genutzt, wobei SysML-Aktivitätsdiagramme sich bzgl. Syntax und Semantik von den UML-Aktivitätsdiagrammen unterscheiden.

1.4.2 Modellierung von Anforderungen vs. Systementwurf

In der Praxis besteht mitunter die Schwierigkeit, Anforderungsdiagramme und Designdiagramme voneinander zu unterscheiden (vgl. z.B [RuQu2012]). Häufig wird die Ursache hierfür in dem Umstand gesehen, dass für die Anforderungsmodellierung die gleichen universellen Modellierungssprachen verwendet werden, wie z.B. UML oder SysML. Tatsächlich liegt die Ursache in den meisten Fällen darin, dass die vermeintlichen Anforderungsdiagramme keine Anforderungen spezifizieren, sondern das Systemdesign, oder dass Anforderungen und Design in Diagrammen vermischt sind. Letzteres ist z.B. dann der Fall, wenn in einem Diagramm das notwendige Verhalten des Systems schon im Hinblick auf einzelne, spezifische Entwurfsentscheidungen modelliert ist, die nicht durch Randbedingungen (Constraints), etwa im Hinblick auf die zu verwendende Technologie, vorgegeben sind (siehe Abschnitt 1.5).

1.4.2.1 Anforderungs- und Entwurfsdiagramme in der Systemanalyse

Im Rahmen der Systemanalyse werden oft Diagramme erstellt, wobei sowohl Entwurfs- als auch Anforderungsdiagramme auftreten. Typischerweise wird im Rahmen der Systemanalyse zunächst ein existierendes System analysiert. Der Betrachtungsgegenstand „System“ kann hier von einzelnen Softwaresystemen bis hin zu komplexen soziotechnischen Systemen reichen, in denen verschiedenste Softwaresysteme und Personen (bzw. Rollen) zusammenwirken, um einen übergeordneten Zweck zu erfüllen, wie es bspw. bei komplexen betrieblichen Informationssystemen der Fall ist. Die Systemanalyse selbst kann aus verschiedenen Perspektiven heraus durchgeführt werden, z.B. funktionszentriert oder datenzentriert (vgl. z.B. [DeMa1979] und [ShMe1988]). Im Rahmen der Systemanalyse wird häufig zunächst das betrachtete System analysiert (z.B. das System im Betrieb und zugehörige Dokumentation) und in Form von Diagrammen, so wie es wahrgenommen wird, modelliert. In diesem Fall wird dann zunächst die technische Inkarnation des Systems modelliert, d.h. die konkrete technische Lösung, so wie sie im Betrieb ist (vgl. [McPa1984]). Das entsprechende Modell der Inkarnation wird dann hinsichtlich der zugrundeliegenden fachlichen Aspekte analysiert, d.h. es wird von der konkreten technischen Umsetzung abstrahiert, um den fachlichen Kern zu identifizieren. Ergebnis dieser Aktivität ist ein Modell der fachlichen Anforderungen des betrachteten Systems. Beide Modelle, sowohl das Inkarnationsmodell (d.h. die technische Lösung) als auch das Modell der fachlichen Anforderungen (auch: Essenzmodell) sind in diesem Falle Ist-Modelle, d.h. Modelle, die bestehende Eigenschaften des betrachteten Systems

dokumentieren. Im Rahmen der Systemanalyse wird dann häufig auf Grundlage des Modells der fachlichen Anforderungen ein Soll-Modell formuliert, das spezifiziert, welche fachlichen Anforderungen von einem neu zu entwickelnden System bzw. im Rahmen eines Änderungsprojektes umzusetzen sind. Diese fachlichen Anforderungen fließen dann wiederum in den Entwicklungsprozess ein. Im Rahmen typischer Systemanalyseprozesse werden demnach sowohl Anforderungsdiagramme als auch Entwurfsdiagramme erstellt, wobei das Ziel der Systemanalyse darin besteht, die fachlichen Anforderungen des betrachteten Systems zu modellieren.

1.4.2.2 Zusammenhang zwischen Anforderungsmodellen und Entwurfsmodellen

Im Zuge der Entwicklung komplexer Softwaresysteme werden Anforderungen und Entwurf häufig sehr stark miteinander verzahnt entwickelt. Diese enge Verzahnung zwischen der Entwicklung von Anforderungen und der Definition einer Lösung in Form eines Systementwurfs wird durch das in Abbildung 3 gezeigte *Twin-Peaks-Modell* verdeutlicht (vgl. hierzu [Nuse2001]).

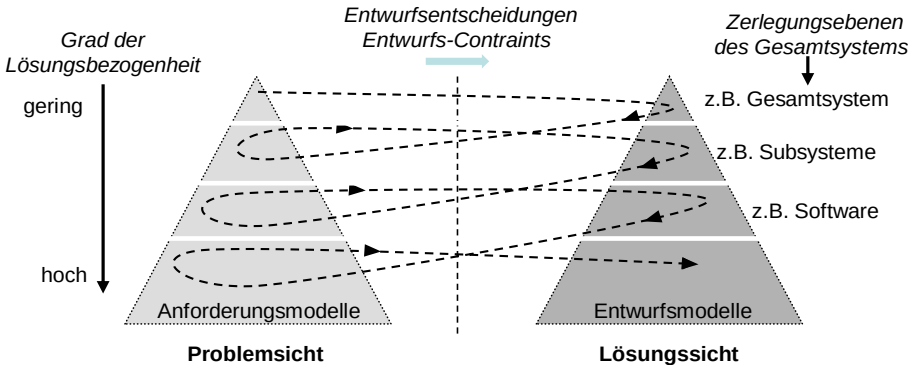

Abbildung 3:Zusammenhang zwischen Anforderungen und Entwurf

Wie in der Abbildung veranschaulicht, besteht bei der Entwicklung komplexer Softwaresysteme eine starke Wechselwirkung zwischen der Definition von Anforderungen und dem Systementwurf. Typischerweise wird zunächst eine Menge eher allgemeiner Anforderungen an das jeweils betrachtete Gesamtsystem erarbeitet, auf deren Basis dann eine erste grobe Systemarchitektur definiert wird, die diesen Anforderungen genügt. Hierzu müssen beim Übergang zwischen Anforderungsdefinition und Systementwurf Entwurfsentscheidungen getroffen und die gegebenen Rahmenbedingungen für den Entwurf (Design-Constraints) eingehalten werden (z.B. die Vorgabe eines zu verwendenden Architekturstils). Ausgehend von der initialen Systemarchitektur, die z.B. aus (logischen) Subsystemen besteht, werden dann z.B. die Anforderungen an die einzelnen Subsysteme spezifiziert. Liegen genügend detaillierte Anforderungen vor wird der initiale Systementwurf wiederum verfeinert. Abbildung 3 illustriert den Zusammenhang zwischen Anforderungen und Entwurf beispielhaft für ein technisches System (Gesamtsystem), bei dem zunächst von der Aufteilung in Hardware- und Software abstrahiert wird. Die Anforderungen an die eigentliche Software des Systems werden hier erst auf der dritten Systemebene spezifiziert. Bei reinen Softwareentwicklungsprojekten wird die zu entwickelnde Software auf der obersten Systemebene eingeordnet. Auf den tieferen Systemebenen werden dann beispielsweise *logische Komponenten* und *Software Parts* betrachtet (siehe hierzu z.B. [ISO26702], [HaHP2001]).

Bei diesem Vorgehen beeinflussen die Entwurfsentscheidungen auf einer Ebene wesentlich die Anforderungsdefinition auf der nächsttieferen Detaillierungsebene, d.h. die Anforderungen der nächsttieferen Ebene beruhen auf den zuvor getroffenen Entwurfsentscheidungen, die wiederum Rahmenbedingungen für die Spezifikation von Anforderungen auf der nächst-

tieferen Ebene sind. Auch wenn es hierbei eine enge Verzahnung zwischen Anforderungen und Architekturentwurf gibt, ist es im Rahmen der Anforderungsmodellierung umso wichtiger, das Anforderungsmodell strikt vom Entwurfsmodell zu trennen und die Beziehungen über geeignete Abhängigkeitsbeziehungen herzustellen (siehe hierzu Abschnitt 1.9). Weitere Details hierzu finden sich z.B. in [Pohl2010], [BDH+2012] und [HaHP2001].

1.5 Sichten der Anforderungsmodellierung

Im Foundation Level des Certified Professional for Requirements Engineering werden drei Sichten bei der Modellierung funktionaler Anforderungen unterschieden (vgl. [PoRu2011]), und zwar (1) die statisch-strukturelle Sicht, (2) die Verhaltenssicht und (3) die Funktionssicht. Aufbauend auf diesen elementaren Sichten der Anforderungsmodellierung wird im Folgenden eine differenziertere Unterscheidung von Sichten in der Anforderungsmodellierung vorgestellt (siehe Abbildung 4).[1]

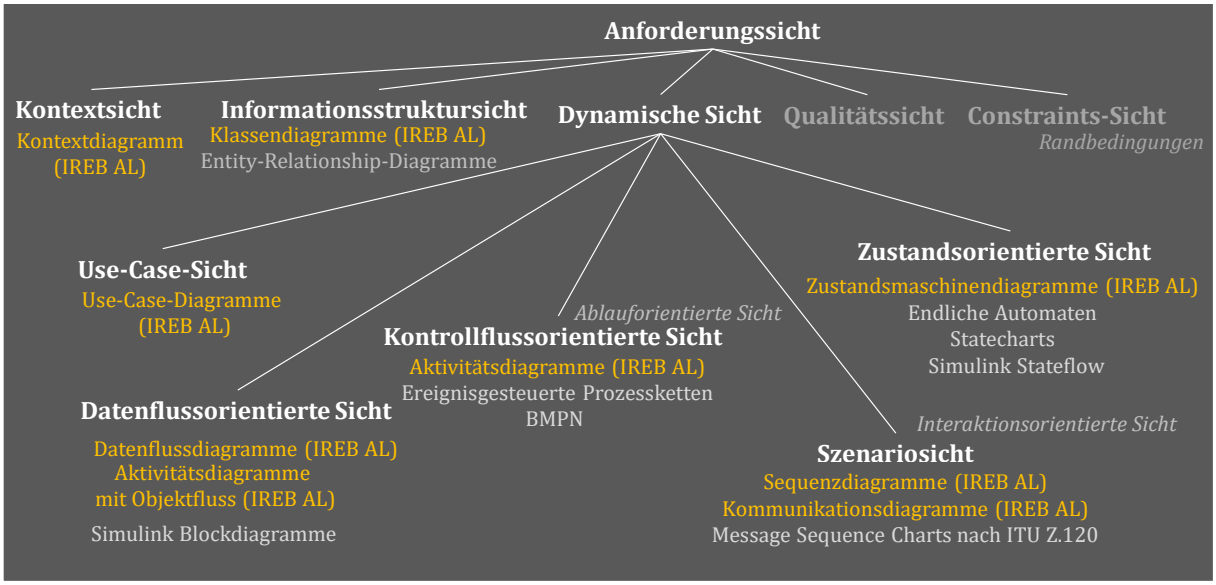

Abbildung 4: Sichten in der Anforderungsmodellierung im IREB Advanced Level Requirements Modeling

1.5.1 Kontextsicht

Eine wesentliche Herausforderung im Requirements Engineering besteht darin, den Kontext des betrachteten Systems (z.B. der zu entwickelnden Software) zu verstehen (vgl. hierzu auch [Weye2010]). Hierzu gehört etwa das Wissen darüber, welche anderen Systeme mit dem betrachteten System in einem operationellen Zusammenhang stehen, Eigenschaften dieser externen Systeme, sowie Wissen darüber, welche Rollen, Personen im Betrieb mit

[1] Sichtenbildung kann auf unterschiedlichste Weise im Rahmen des Requirements Engineering etabliert werden. Beispielsweise können auch Sichten definiert werden, die spezifische Belange („Concerns“) von Stakeholdern adressieren. So kann z.B. eine „Nutzersicht“ auf die Anforderungen des Systems definiert werden, in der lediglich Anforderungen betrachtet (modelliert) werden, die unmittelbar die Nutzung des betrachteten Systems betreffen. In einer „Wartungsingenieursicht“ würden dann nur solche Anforderungen des Systems betrachtet, die unmittelbar die Wartung des Systems im Betrieb betreffen. Verschiedene „Philosophien“ der Sichtenbildung können auch kombiniert eingesetzt werden, um den Umfang und die Komplexität der Anforderungsmodellierung zu beherrschen. So ist beispielsweise denkbar, dass die „Nutzersicht“ und „Wartungsingenieursicht“ jeweils aus einer „Informationsstruktursicht“ und einer „dynamischen Sicht“ betrachtet werden. Über gemeinsame Konzepte bzw. Abbildungsbeziehungen sind die Anforderungsmodelle der verschiedenen Sichten dann zu einen „Gesamtmodell“ integrierbar.

dem System interagieren und welche für das System relevanten Eigenschaften diese besitzen. Typischerweise dient die Kontextmodellierung vor allem auch dazu, die notwendigen Schnittstellen zwischen dem betrachteten Systems und seinem Kontext zu identifizieren.

1.5.2 Informationsstruktursicht

Die Informationsstruktursicht fokussiert auf Anforderungen des betrachteten Systems, die sich auf statisch-strukturelle Aspekte der Funktionalität beziehen, wie z.B. die Struktur von Daten, die vom System verarbeitet werden müssen. Typische Diagrammtypen, die hier eingesetzt werden, sind Klassendiagramme oder verschiedene Dialekte der Entity-Relationship-Diagramme (z.B. nach Chen oder im FMC-Ansatz).

1.5.3 Dynamische Sicht

Die dynamische Sicht fokussiert auf solche Anforderungen des betrachteten Systems, die sich auf dynamische Aspekte der Funktionalität beziehen (vgl. z.B [RuQu2012]). Im Sinne des Foundation Level des Certified Professional for Requirements Engineering wird die dynamische Sicht auf die Anforderungen eines Systems durch die Verhaltens- und Funktionssicht gebildet. Zur Modellierung der Anforderungen in der dynamischen Sicht wird im Advanced Level Requirements Modeling die dynamische Sicht stärker differenziert (siehe Abschnitt 1.6). Typische Diagrammtypen, die hier zur Anforderungsmodellierung eingesetzt werden, sind z.B. Use-Case-Diagramme, Aktivitätsdiagramme, Zustandsmaschinendiagramme, Datenflussdiagramme und Sequenzdiagramme.

1.5.4 Qualitätssicht

Die Qualitätssicht fokussiert auf solche Anforderungen des betrachteten Systems, die sich auf notwendige Qualitäten des betrachteten Systems bzw. einzelner Systembestandteile beziehen. Auch wenn in der Forschung eine Reihe von Ansätzen zur modellbasierten Spezifikation von Qualitätsanforderungen existiert (z.B. [HKDW2012]), werden in der Praxis bis heute Qualitätsanforderungen (wie z.B. Anforderungen bzgl. Performance, Verlässlichkeit, Realzeitverhalten, Safety oder Robustheit) innerhalb von Anforderungsmodellen hauptsächlich durch textuelle Ergänzungen bzw. als Annotation an spezifische Modellelemente in Anforderungsdiagrammen spezifiziert (vgl. z.B. [RiWe2007]). Eine ausführliche Taxonomie für Anforderungen in der Qualitätssicht (Qualitätsanforderungen) findet sich in ISO 25010 [ISO25010]. Detaillierte Informationen zur Dokumentation von Anforderungen der Qualitätssicht finden sich z.B. in [Pohl2010], [Hrus2014] und [Rupp2007].

1.5.5 Constraints-Sicht

Die Constraints-Sicht fokussiert auf Anforderungen im Sinne von Randbedingungen (d.h. externe Einschränkungen), die von dem zu entwickelnden System (oder dem zugehörigen Entwicklungsprozess) einzuhalten sind (vgl. [ISO29148]). Typische Randbedingungen sind dabei z.B. organisatorischer, normativer oder technologischer Natur. Technologische Randbedingungen treten z.B in Form von Design-Constraints auf, durch die etwa ein spezifischer Architekturstil (z.B. servicebasiert oder Client-Server) für das zu entwickelnde System vorgeschrieben wird. Solche Randbedingungen werden häufig in textueller Form (bzw. durch textuelle Ergänzungen in Anforderungsmodellen) dokumentiert, wohingegen sich z.B. zur Dokumentation organisatorischer oder technischer Randbedingungen häufig auch spezifische Diagrammtypen wie Klassen- oder Komponentendiagramme eignen. Detaillierte Informationen zu Randbedingungen finden sich z.B. in [RoRo2006].

1.6 Sichten der dynamischen Sicht in der Anforderungsmodellierung

Die dynamische Sicht in der Modellierung von Anforderungen betrachtet solche Anforderungen, die sich auf die zeitlich-logischen Zusammenhänge im notwendigen Verhalten des Systems beziehen. Heutige betriebliche Informationssysteme und mehr noch intelligente eingebettete Systeme besitzen ein sehr umfangreiches und komplexes Gebilde solcher Zusammenhänge, die im Rahmen des Requirements Engineering aufgedeckt, analysiert und in den Anforderungen spezifiziert werden müssen. Um den Umfang und die Komplexität solcher dynamischer Zusammenhänge im Systemverhalten innerhalb der Anforderungsmodellierung handhabbar zu machen, wird die dynamische Sicht wiederum in Sichten unterteilt, deren Integration zu einem Gesamtmodell der dynamischen Sicht auf die Anforderungen des betrachteten Systems führt. Wie in Abbildung 4 gezeigt, wird die dynamische Sicht auf die Anforderungen eines Systems wiederum in verschiedene Sichten unterteilt.

1.6.1 Use-Case-Sicht (Nutzerfunktionen und Abhängigkeiten zum Systemkontext)

In der Use-Case-Sicht innerhalb der dynamischen Sicht werden die groben Nutzerfunktionen des Systems und deren Beziehungen zu Akteuren im Systemkontext betrachtet. Eine grobe Nutzerfunktion charakterisiert eine Funktionalität, die das System zur Verfügung stellen muss, damit ein Akteur im Kontext einen Nutzen (Mehrwert) erfährt. Typischerweise werden hier zur Modellierung Use-Case-Diagramme eingesetzt.

1.6.2 Datenflussorientierte Sicht (Systemfunktionen und Datenabhängigkeiten)

In der Datenflussorientierten Sicht innerhalb der dynamischen Sicht werden die an der Schnittstelle des Systems wahrnehmbaren Funktionen und deren Datenabhängigkeiten untereinander und zu Akteuren im Systemkontext betrachtet. Die Funktionen können dabei auf verschiedenen Granularitätsstufen betrachtet werden, z.B. von groben Nutzerfunktionen (z.B. Use-Cases) bis hin zu z.B. feingranularen fachlichen Funktionen, deren Zusammenwirken die Funktionalität des Use-Cases realisiert. Typische Diagrammtypen, die hier zum Einsatz kommen, sind Datenflussdiagramme (z.B. nach DeMarco [DeMa1979]) und auch Aktivitätsdiagramme, die den Objektfluss zwischen Aktionen fokussieren.

1.6.3 Kontrollflussorientierte Sicht (Ablauflogik von Prozessen)

In der Kontrollflussorientierten Sicht innerhalb der dynamischen Sicht werden die an der Schnittstelle des Systems wahrnehmbaren Prozesse (bzw. Aktivitäten oder Aktionen) und deren Ablauflogik betrachtet. Die Kontrollflussbeziehungen werden in Prozessen betrachtet, die z.B. in Form sequentieller, alternativer oder nebenläufiger Abfolgen auftreten. Typischerweise kommen zur Modellierung in der kontrollflussorientierten Sicht UML- bzw. SysML-Aktivitätsdiagramme zum Einsatz. Speziell bei der Business-Analyse findet man zur Modellierung auf der Ebene von Geschäftsprozessen auch (erweiterte) Ereignisgesteuerte Prozessketten oder BPMN-Diagramme.

1.6.4 Zustandsorientierte Sicht (Zustände und Zustandswechsel)

In der Zustandsorientierten Sicht innerhalb der dynamischen Sicht wird der benötigte Zustandsraum des Systems insbesondere im Hinblick auf das reaktive Verhalten des Systems gegenüber seinem Systemkontext modelliert. In dieser Sicht werden die an der Schnittstelle des betrachteten Systems gegenüber dem Systemkontext wahrnehmbare Zustände und Zustandswechsel modelliert. Ein Zustandswechsel des betrachteten Systems kann dabei durch ein Ereignis aus dem Systemkontext, durch ein Zeitereignis oder ein systeminternes Ereignis ausgelöst werden. Typischerweise kommen hier Endliche Automaten, Harel Statecharts oder die auf diesen Konzepten basierenden Zustandsmaschinendiagramme der UML zum Einsatz.

1.6.5 Szenariosicht (Interaktionssequenzen zwischen Akteuren und dem System)

In der Szenariosicht innerhalb der dynamischen Sicht werden Interaktionen zwischen Akteuren im Systemkontext und dem System betrachtet, durch die ein oder mehrere Akteure im Systemkontext einen Mehrwert erhalten bzw. ein Ziel erreichen (z.B. Bargeldversorgung durch die Nutzung eines Geldautomaten). Szenarien werden dabei häufig zur Konkretisierung der Use-Cases in Use-Case-Diagrammen eingesetzt, indem mittels Szenarien beschrieben wird, durch welche Interaktion zwischen dem System und Akteuren im Systemkontext der Use-Case erfolgreich ausgeführt wird. Typischerweise wird in der Szenariomodellierung nicht nur die unmittelbare Interaktion zwischen Akteuren und dem betrachteten System modelliert, sondern auch der Nachrichtenaustausch zwischen Akteuren im Kontext des Systems. Zur Modellierung von Szenarien werden typischerweise Sequenzdiagramme der UML/SysML oder Message Sequence Charts nach dem ITU Standards Z.120 [ITU2004] eingesetzt.

1.7 Anpassung von Modellierungssprachen zur Anforderungsmodellierung

Die UML und SysML besitzen ein Konzept, um die verschiedenen Modellierungssprachen anpassen oder erweitern zu können. Dies ist beispielsweise dann sinnvoll, wenn spezifische Konzepte eines Projektes oder einer Anwendungsdomäne in der Sprachbasis verankert werden sollen. Die Anpassung der UML/SysML geschieht typischerweise über die Definition von Stereotypen, mit denen Notationselementen eine spezielle Bedeutung (oder Semantik) gegeben werden kann. In der UML und SysML können sämtliche Notationselemente durch Stereotypen angepasst oder erweitert werden. Die Definition eines Stereotypen besteht aus einem syntaktischen Teil, in dem die Repräsentation des Stereotypen und die gewünschten Bezüge zu Notationselementen festgelegt werden, sowie aus einen semantischen Teil, der die Bedeutung des Stereotyps festlegt. Stereotypen werden in Diagrammen der UML/SysML in Form spitzer Klammern modelliert. So könnte z.B. durch den Stereotyp <<Domäne>> für Klassen innerhalb eines Klassendiagramms (⮜ Definition der Syntax des Stereotyps) ausgedrückt werden, dass es sich bei den entsprechenden Klassen, die diesen Stereotypen aufweisen, um Klassen handelt, die spezifisch für die jeweilige Anwendungsdomäne sind und deren fachliche Bedeutung innerhalb eines Domänenglossars genauer definiert ist (⮜ Definition der Semantik des Stereotyps).

1.8 Integration textueller Anforderungen in das Anforderungsmodell

Die SysML besitzt gegenüber der UML ein spezielles Notationsmittel, um textuelle Anforderungen zu modellieren. Zusätzlich ist ein spezieller Diagrammtyp, das *Anforderungsdiagramm*, definiert, welches weder der Struktur- noch der Verhaltenssicht zugeordnet ist. Dieser Diagrammtyp gestattet es, Beziehungen zwischen textuellen Anforderungen zu modellieren oder textuell spezifizierte Anforderungen an Modellelemente von SysML-Diagrammen anzuhängen bzw. diese zu referenzieren. Diese Art der „Modellierung" textueller Anforderungen wird häufig eingesetzt, um vorgegebene Anforderungen (z.B. aus einer Fachbereichssicht) in das Anforderungsmodell einzubinden, wobei eine solche Einbindung im Wesentlichen dazu dient, die modellierten Anforderungen mit den vorgebebenen textuellen Anforderungen in Beziehung zu setzen. Damit kann z.B. ausgedrückt werden, durch welche modellierten Anforderungen eine textuelle Anforderung konkretisiert wird.

Die meisten marktverfügbaren UML-Werkzeuge bieten allerdings bereits die Möglichkeit, textuelle Anforderungen in beliebigen Diagrammtypen, und nicht nur in *Anforderungsdiagrammen* („Requirements Diagramm") zu verwenden. Damit bietet sich z.B. die Möglichkeit, Anforderungen als Alternative zur diagrammatischen Spezifikation textuell zu spezifizieren, weil z.B. nach Meinung des Requirements Engineers bestimmte Anforderungen zweckdienlicher als textuelle Anforderungen spezifiziert sein sollten. So kann z.B. eine Aktion in einem Ablauf durch eine Menge textueller Anforderungen verfeinert werden, die innerhalb des Anforderungsmodells dann zu dieser Aktion in Bezug gesetzt sind (z.B. durch eine entsprechende Tracing-Beziehung). Durch dieses Konzept der Integration textuell spezifizierter Anforderungen innerhalb von Anforderungsmodellen können z.B. auch Qualitätsanforderungen, die sich auf eine bestimmte Aktion beziehen (z.B. Anforderungen bzgl. der Performance dieser Aktion), als textuelle Anforderungen spezifiziert werden, indem diese textuellen Anforderungen innerhalb des Diagramms, in dem die Aktion modelliert wurde, mit dieser Aktion in Beziehung gesetzt werden. Durch dieses Konzept des ergänzenden Einsatzes textueller Anforderungen können die Modellelemente aus den verschiedenen Diagrammtypen (und damit die entsprechenden Diagramme) zur Anforderungsmodellierung erweitert werden, um textuelle Anforderungen mit Anforderungsdiagrammen innerhalb des Anforderungsmodells in Beziehung zu setzen.

1.9 Dokumentation von Abhängigkeiten zwischen Modellelementen

Unabhängig davon, ob Anforderungen in Form von Anforderungsdiagrammen oder textuell vorliegen, können diese im Zuge der modellbasierten Dokumentation von Anforderungen mit UML/SysML mit Hilfe explizit definierter Abhängigkeitsbeziehungen zueinander in Bezug gesetzt werden. Hierzu werden geeignete Stereotypen für Abhängigkeitsbeziehungen zwischen Modellelementen des Anforderungsmodells definiert (siehe auch Abschnitt 1.7). Die Art der zu verwendenden Stereotypen, d.h. deren Syntax und Semantik, hängt dabei in vielen Fällen stark vom Projektkontext und der Anwendungsdomäne ab, so dass in einem Entwicklungsprojekt mit den Projektbeteiligten festgelegt werden muss, welche Abhängigkeitstypen zwischen Anforderungen benötigt werden (siehe hierzu auch [RaJa2001]). Anschließend müssen die benötigten Abhängigkeitsbeziehungen in den entsprechenden Werkzeugen definiert werden. Typische Beispiele für häufig vorzufindende Abhängigkeitsbeziehungen zwischen Modellelementen innerhalb eines Anforderungsmodells sind:

- <<refines>>: A <<refines>> B drückt aus, dass eine einzelne Anforderung oder eine Menge von Anforderungen A eine einzelne Anforderung oder eine Menge von Anforderung B verfeinert, indem z.B. A eine oder mehrere zusätzliche Anforderungen zu den Anforderungen B spezifiziert.
- <<realizes>>: A <<realize>> B drückt aus, dass die Anforderungen in A die Anforderungen B realisieren. Dies wird z.B. dann verwendet, wenn A die Anforderungen an eine Komponente sind, die dazu führt, dass die Anforderungen B des Gesamtsystems erfüllt werden. Diese Art des Tracings basiert dabei allerdings darauf, dass entweder im Entwicklungsprozess schon Design-Entscheidungen über die Struktur der Lösung getroffen wurden, oder die Notwendigkeit einer solche Komponente bzw. Vorgaben über die Strukturierung des Gesamtsystems in Komponenten bereits als Randbedingungen für das Requirements Engineering bestehen (vgl. hierzu z.B. [BDH+2012]).
- <<satisfies>>: A <<satisfies>> B drückt aus, dass eine einzelne Anforderung oder eine Menge von Anforderungen A eine einzelne oder eine Menge von Anforderungen B erfüllt. Dieser Typ von Abhängigkeitsbeziehungen wird z.B. in Auftraggeber-Auftragnehmer-Beziehungen eingesetzt, wenn detailliertere Anforderungen, die durch den Auftragnehmer spezifiziert wurden, mit generelleren Anforderungen des Auftraggebers in Bezug gesetzt werden sollen, um auszudrücken, dass die Anforderungen A des Auftragnehmers die Anforderungen B des Auftraggebers erfüllen. Typischerweise lassen sich durch solche Abhängigkeiten auch Beziehungen zwischen Anforderungen im Pflichtenheft und Anforderungen im Lastenheft ausdrücken, um z.B. den Nachweis zu unterstützen, dass die für das betrachtete System im Pflichtenheft spezifizierten Anforderungen dazu führen, dass das realisierte System die Anforderungen im Lastenheft erfüllt. Der Abhängigkeitstyp <<satisfies>> besitzt eine gewisse Ähnlichkeit mit dem Abhängigkeitstyp <<realizes>>, wobei Abhängigkeiten des Typs <<satisfies>> typischerweise an der Schnittstelle zwischen Auftraggeber und Auftragnehmer verwendet werden.

1.10 Vorteile der Anforderungsmodellierung

Die Spezifikation von Anforderungen durch Diagramme besitzt im Vergleich zur textuellen Spezifikation von Anforderungen eine Reihe wesentlicher Vorteile, wie z.B.:

- *Bessere Verständlichkeit der Anforderungen:* Im Allgemeinen hat sich in der Kognitionsforschung gezeigt, dass in Diagrammen visualisiert Sachverhalte für den Menschen besser verständlich und einfacher zu merken sind als entsprechende textuelle Beschreibungen dieser Sacherhalte (vgl. [LaSi1987]). Im Speziellen bedeutet dies, dass Anforderungen, die durch Diagramme spezifiziert sind, einfacher verstanden und memorisiert werden können als Anforderungen, die in textueller Form vorliegen. *„Ein Bild sagt mehr als tausend Worte!“*
- *Inhärente Unterstützung des Prinzips „Trennung von Belangen“:* Diagrammtypen sind für einen bestimmten Verwendungszweck definiert und zwingen den Modellierer durch die verfügbaren Notationselemente (Semantik) und die in der Sprache zugelassene Art und Weise, wie diese Notationselemente kombiniert werden dürfen (Syntax), dazu, sich auf einen Sachverhalt zu konzentrieren. So sollten Zustandsmaschinendiagramme im Rahmen der Anforderungsmodellierung z.B. für die Modellierung des notwendigen reaktiven Verhaltens des betrachteten Systems eingesetzt werden und beispielsweise nicht für die Modellierung von Abläufen oder Datenstrukturen. Die Trennung von Belangen wird in der Anforderungsmodellierung durch verschiedene Sichten

etabliert, wobei die Anforderungsmodelle der einzelnen Sichten durch gemeinsame Konzepte integriert werden können. Hierdurch können Aussagen über verschiedene Anforderungssichten hinweg getroffen werden. Detaillierte Informationen hierzu finden sich u.a. in [DaTW2012].

- *Inhärente Unterstützung des Prinzips „Teile und Herrsche“:* Durch die Verwendung verschiedener Diagrammtypen können zunächst isoliert jeweils die spezifischen Anforderungen modelliert werden, die von dem jeweiligen Diagrammtyp unterstützt werden. Die Diagramme unterschiedlichen Typs können dann über gemeinsame Konzepte bzw. definierte Mapping-Beziehungen zusammengeführt werden, um ein integriertes Anforderungsmodell zu erhalten. Diese Eigenschaft der diagrammatischen Spezifikation von Anforderungen unterstützt den Requirements Engineer z.B. dabei, das Gesamtproblem, d.h. die Spezifikation von Anforderungen eines Systems, in beherrschbare Teilprobleme (z.B. die Spezifikation der Anforderungen eines Subsystems) zu zerlegen. Die Zusammenführung der einzelnen Anforderungsmodelle der Teilprobleme bildet dann das Anforderungsmodell des jeweils übergeordneten Systems. Detailliertere Informationen hierzu finden sich u.a. in [BDH+2012] und [HaHP2001].
- *Geringeres Risiko für Mehrdeutigkeiten:* Durch den im Vergleich zur natürlichen Sprachen höheren Formalisierungsgrad der Modellierungssprachen zur Anforderungsmodellierung haben diagrammatisch spezifizierte Anforderungen ein geringes Risiko der Mehrdeutigkeit bzw. Fehlinterpretation durch andere am Entwicklungsprozess beteiligte Personen (z.B. dem Architekten, Entwickler, Tester).
- *Höheres Potenzial zur maschinellen Analyse der Anforderungen*: Durch den im Vergleich zur textuellen Spezifikation von Anforderungen höheren Formalisierungsgrad von diagrammatisch spezifizierten Anforderungen können solche Anforderungen in größerem Umfang oder gar vollständig maschinell analysiert werden (z.B. eine Analyse der Erreichbarkeit von Zuständen in einem Anforderungsdiagramm der zustandsbasierten Sicht).
- *Höheres Potenzial zur maschinellen Weiterverarbeitung der Anforderungen*: Der höhere Formalisierungsgrad diagrammatischer Anforderungen erhöht auch die Möglichkeit, die Anforderungen des Systems maschinell weiterzuverarbeiten und diese in anderen Entwicklungsdisziplinen zu nutzen, wie z.B. zur Ableitung von Testfällen für den Systemtest aus Anforderungsdiagrammen der kontrollflussorientierten Sicht.
- *Anforderungen im Kontext:* Die Modellierung von Anforderungen führt dazu, dass einzelne Modellelemente innerhalb des Anforderungsmodells (siehe Abschnitt 1.3) und die Beziehungen einzelner Anforderungen zu anderen Anforderungen unmittelbar im Anforderungsmodell repräsentiert sind. Dies erleichtert den Umgang mit umfangreichen und komplexen Anforderungen und fördert das Verständnis der Anforderungen dadurch, dass für den Leser der Anforderungen der Kontext einer Anforderung im Anforderungsmodell ersichtlich ist. Beispielsweise ist in einem Aktivitätsdiagramm für jede Aktion unmittelbar ersichtlich, mit welchen anderen Aktionen diese in einem Zusammenhang steht und welche Zustandsänderung des betrachteten Systems durch die Ausführung der Aktion ausgelöst wird.

1.11 Qualität von Anforderungsmodellen

Die Qualität eines Anforderungsmodells basiert auf der Qualität seiner Bestandteile. Wie in Abschnitt 1.1. dargestellt, setzt sich das Anforderungsmodell eines Systems aus einer Menge

von Diagrammen und textuellen Ergänzungen zusammen. Werden Anforderungen modelliert, ist ein wesentlicher Teil der Anforderungen in den Diagrammen spezifiziert, so dass die Qualität des Anforderungsmodells sich zu wesentlichen Teilen aus der Qualität der einzelnen Diagramme und deren Beziehungen zueinander ergibt. Die Qualität der einzelnen Diagramme wiederum bestimmt sich durch die Qualität der Modellelemente innerhalb der Diagramme und der zugehörigen textuellen Ergänzungen. Der linke Bereich von Abbildung 5 verdeutlicht die hierarchische Struktur der Beurteilung der Qualität von Anforderungsmodellen.

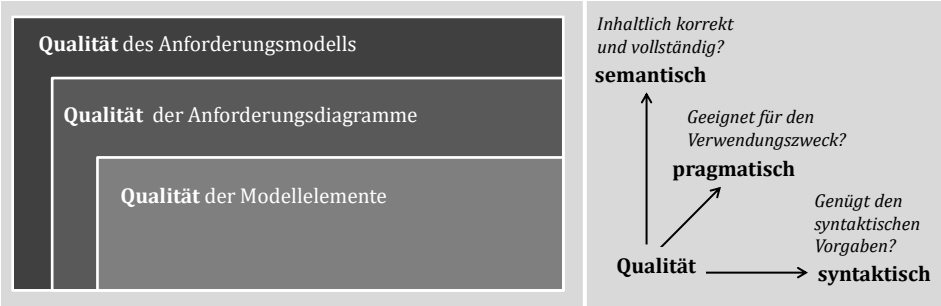

Abbildung 5: Beurteilung der Qualität von Anforderungsmodellen

Die Qualität des Anforderungsmodells, der Anforderungsdiagramme und der Modellelemente kann anhand dreier Kriterien beurteilt werden (vgl. z.B. [LiSS1997]):

Syntaktische Qualität

Die syntaktische Qualität drückt aus, inwieweit ein einzelnes Modellelement (grafisch oder textuell), ein Anforderungsdiagramm oder ein Anforderungsmodell den geltenden syntaktischen Vorgaben genügt. Soll beispielsweise die syntaktische Qualität eines Anforderungsdiagramms der Szenariosicht (das in Form eines Sequenzdiagramms der UML vorliegt) beurteilt werden, muss geprüft werden, inwieweit dieses Diagramm die syntaktischen Vorgaben der UML einhält. Beispielsweise schreibt die Syntax der Sequenzdiagramme vor, dass eine synchrone Nachricht auf einer bestimmten Detaillierungsebene aus einem Funktionsaufruf und einer Antwortnachricht besteht. Tritt in einem durch ein Sequenzdiagramm modellierten Szenario lediglich eine Antwortnachricht ohne einen vorausgegangenen Funktionsaufruf auf, so genügt dies nicht den syntaktischen Vorgaben der zugrundeliegenden Modellierungssprache und reduziert damit die syntaktische Qualität des Diagramms. Werden einschlägige Modellierungswerkzeuge zur Modellierung von Anforderungen eingesetzt, so wird die syntaktische Qualität der erstellten Diagramme in der Regel durch das Werkzeug sichergestellt.

Semantische Qualität

Die semantische Qualität drückt aus, inwieweit das einzelne Modellelement (grafisch oder textuell), das Anforderungsdiagramm oder das Anforderungsmodell den jeweils repräsentierten Sachverhalt vollständig und korrekt abbildet. Beispielshaft sei angenommen, dass nach dem Einführen der EC-Karte in den Kartenschlitz eines Geldautomaten zunächst die PIN des Kunden erfragt werden soll. Ist in einem entsprechenden Anforderungsdiagramm der kontrollflussorientierten Sicht (z.B. einem Aktivitätsdiagramm) modelliert, dass nach dem Einlesen der Kartendaten zunächst die Höhe des gewünschten Auszahlungsbetrags vom Kunden erfragt wird, stellt dies einen semantischen Defekt im entsprechenden Diagramm dar, da der tatsächlich geforderte Ablauf davon abweicht. Ein solcher Defekt in einem Anforderungsdiagramm beeinträchtigt die semantische Qualität des übergeordneten Anforderungsmodells.

Pragmatische Qualität

Die pragmatische Qualität drückt aus, inwieweit das einzelne Modellelement (grafisch oder textuell), das Anforderungsdiagramm oder das Anforderungsmodell für den Verwendungszweck geeignet ist. Hier stellt sich insbesondere die Frage danach, ob der Detaillierungs- bzw. Abstraktionsgrad für den Verwendungszweck angemessen ist. Für ein einzelnes Modellelement bedeutet dies etwa, ob das Modellelement (wie z.B. ein Zustandsübergang in einem zustandsorientierten Anforderungsmodell) im richtigen Detaillierungsgrad spezifiziert ist (z.B. Ist nur das auslösende Ereignis angegeben? Oder sind auch die zusätzlich geltenden Bedingungen für den Zustandswechsel und ein auszulösendes Verhalten angegeben?). Die pragmatische Qualität eines einzelnen Modellelements, eines Anforderungsdiagramms oder eines Anforderungsmodells kann nur beurteilt werden, wenn der Adressat und der Zweck des Diagramms bekannt sind. Da die Pragmatik bestimmt, welche Abstraktionen sinnvoll sind, hat dies auch eine unmittelbare Auswirkung auf die Beurteilung der semantischen Qualität, d.h. die Vollständigkeit eines Modellelements, eines Anforderungsdiagramms oder eines Anforderungsmodells kann nur im Hinblick auf die aus pragmatischer Sicht sinnvollen Abstraktionen beurteilt werden.

1.12 Vertiefende Literatur

Terminologie in der Anforderungsmodellierung

- Glinz, M.: A Glossary of Requirements Engineering Terminology. Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 1.1, May 2011.

Anforderungsmodellierung

- Pohl, K.: Requirements Engineering – Fundaments, Principles, Techniques. Springer, 2010.
- Rupp, C.; Queins, S.: UML 2 Glasklar – Praxiswissen für die UML-Modellierung. Carl Hanser Verlag 2012.
- Daun, M.; Tenbergen, B.; Weyer, T.: Requirements Viewpoint. In: Pohl, K.; Hönninger, H.; Achatz, R.; Broy, M.: Model-based Engineering of Embedded Systems, Springer, Heidelberg 2012.
- Davis, A. M.: Software Requirements – Objects, Functions, States. 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

Qualität von Anforderungsmodellen

- Lindland, O. I.; Sindre, G.; Sølverg, A.: Understanding Quality in Conceptual Modeling. IEEE Software, Vol. 22, Nr. 2, IEEE Press, 1994, 42-49.
- Pohl, K.: Requirements Engineering – Fundaments, Principles, Techniques. Springer, 2010.

2 Kontextmodellierung

Eine große Herausforderung im Requirements Engineering besteht darin, den Kontext des betrachteten Systems zu verstehen. Je komplexer und kritischer das zu entwickelnde System ist, umso wichtiger ist es, das Wissen über dessen Kontext zweckmäßig zu dokumentieren. Hierzu gehört etwa Wissen darüber, welche anderen Systeme mit dem betrachteten System im Betrieb in einem operationellen Zusammenhang stehen, Eigenschaften dieser externen Systeme sowie Wissen darüber, welche Rollen bzw. Personen im Betrieb mit dem System interagieren und welche für das System relevanten Eigenschaften diese besitzen. Zudem dient die Kontextmodellierung unter anderem auch dazu, die notwendigen Schnittstellen des betrachteten Systems zu identifizieren.

2.1 Zweck

Im Requirements Engineering wird der Scope des betrachteten Systems festgelegt (d.h. die Systemgrenze bestimmt) und dabei auch die Abgrenzung des betrachteten Systems gegenüber seinem Kontext vorgenommen. Hierzu müssen u.a. die Einflüsse des Kontexts auf das System untersucht und idealerweise auch dokumentiert werden. Je komplexer und kritischer das zu entwickelnde System ist, umso wichtiger ist es, das Wissen über den Kontext zweckmäßig zu dokumentieren. Hierzu gehört etwa das Wissen darüber,

- welche Rollen und Personen im Betrieb mit dem System interagieren,
- welche anderen Systeme mit dem betrachteten System im Betrieb in einem operationellen Zusammenhang stehen,
- wie die Schnittstelle zwischen dem betrachteten System und den Personen und Systemen im Kontext beschaffen ist.

Des Weiteren kann die Kontextsicht helfen, über Eigenschaften dieser externen Systeme (Funktionen, Qualitäten) nachzudenken, welche für das System relevant sind.

Die Kontextsicht dokumentiert Eigenschaften des Systemkontexts. Demgegenüber werden in den folgenden Kapiteln hauptsächlich die im Scope „wahrnehmbaren“ notwendigen Eigenschaften des Systems spezifiziert, die dieses aufweisen muss, damit es im Betrieb seinen Zweck erfüllt (u.a. die Ziele der Stakeholder erfüllt und dabei alle Rahmenbedingungen einhält). Die Kontextsicht dokumentiert somit einen wesentlichen Aspekt der Arbeit eines Requirements Engineers, wenn er die Schnittstelle zwischen System und Kontext festlegt.

2.2 Kontextdiagramme

Die Kontextsicht dient aus Anforderungssicht dazu, den Scope eines Systems festzulegen, d.h. die Grenze zwischen Funktionalität **im** und **außerhalb** des Scopes zu ziehen. Als Darstellungsmittel dafür dient oft das klassische Kontextdiagramm der Strukturierten Analyse (SA) [DeMa1979], das jedoch heute – da es kaum mehr Werkzeuge zur Unterstützung von SA gibt – durch zahlreiche andere Diagrammtypen inhaltlich äquivalent ausgedrückt werden kann (z.B. durch ein UML-Klassendiagramm, ein Use-Case-Diagramm oder ein Komponentendiagramm). Ebenso kann als Ersatz für ein Kontextdiagramm eine tabellarische Darstellung verwendet werden, solange die im Folgenden aufgeführten Grundelemente erhalten bleiben.

2.2.1 Grundelemente von Kontextdiagrammen

Die wesentlichen drei Grundelemente eines Kontextdiagramms sind:

- das betrachtete System (genauer: die Systemgrenze);
- die Nachbarsysteme bzw. Akteure des betrachteten Systems (alle Menschen, Rollen, IT-Systeme, Geräte, etc. zu denen das System Schnittstellen hat);
- die (logischen) Schnittstellen zwischen dem System und den Nachbarsystemen.

Da erfahrungsgemäß die Schnittstellen zwischen System und Kontext am besten über die ein- und ausgehenden Daten festgelegt werden kann, konzentriert sich das klassische Kontextdiagramm auf diese benannten Ein- und Ausgabedaten von und zu den Nachbarsystemen. In diesem Sinne ist das Kontextdiagramm die abstrakteste Form eines Datenflussdiagramms (siehe Kapitel 4.3), in dem die komplette Funktionalität des Systems bewusst auf eine Funktion (nämlich das Gesamtsystem) abstrahiert wird. Der Schwerpunkt dieses Diagramms liegt auf der Identifikation aller Schnittstellen des betrachteten Systems.

2.2.2 Beispiel für ein Kontextdiagramm

Abbildung 6 zeigt ein Beispiel eines Kontextdiagramms der Struktuierten Analyse. Das Gesamtsystem (hier: ein Frühwarnsystem im Bergbau) wird als Kreis in der Mitte dargestellt. Die menschlichen Nachbarsysteme wurden in dem Beispiel als Strichmännchen dargestellt, die organisatorischen und technischen Nachbarsysteme als Kästchen. Die Schnittstelle ist in Form von benannten Datenflüssen von und zu den Nachbarsystemen modelliert.

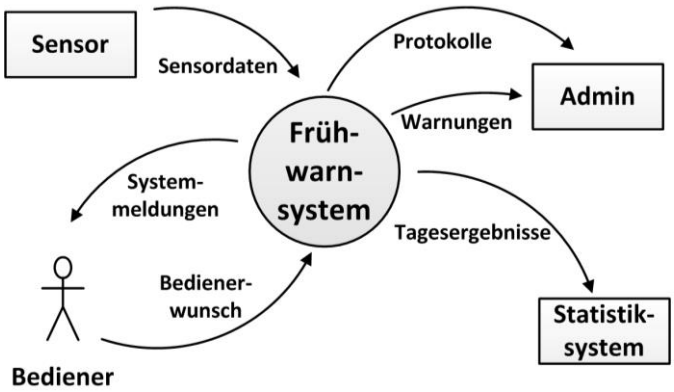

Abbildung 6: Beispiel eines Kontextdiagramms

Zur Modellierung des Systemkontexts werden heute z.B. auch Blockdigramme der SysML [OMG2010a] eingesetzt. Abbildung 7 zeigt das Kontextdiagramm einer automatisierten Maschine zur Herstellung von Zylinderköpfen für Fahrzeuge (vgl. [DaTW2012]).

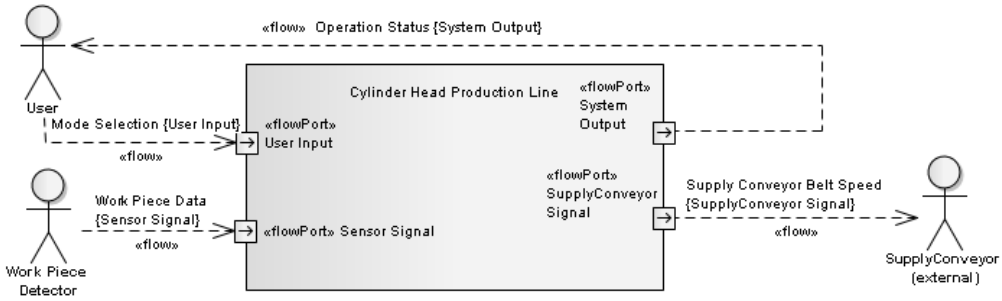

Abbildung 7: Beispiel für ein Kontextdiagramm in Form eines Blockdiagramms der SysML

Das Diagramm zeigt Akteure im Systemkontext sowie die Datenflüsse zwischen Akteuren und dem betrachteten System. Solche Kontextdiagramme auf Basis der SysML sind in Bezug auf die Informationen, die sie über den Systemkontext dokumentieren, sehr ähnlich den Kontextdiagrammen, die auf den Datenflussdiagrammen der Strukturierten Analyse beruhen.

2.2.3 Notationselemente für die Modellierung von Kontextdiagrammen mit Datenflussdiagrammen

Zur Modellierung datenflussorientierter Kontextdiagramme können z.B. Datenflussdiagramme verwendet werden. Abbildung 8 zeigt mögliche Modellierungskonstrukte zur Konstruktion datenflussorientierter Kontextdiagramme auf Basis von Datenflussdiagrammen nach DeMarco (vgl. [DeMa1979]).

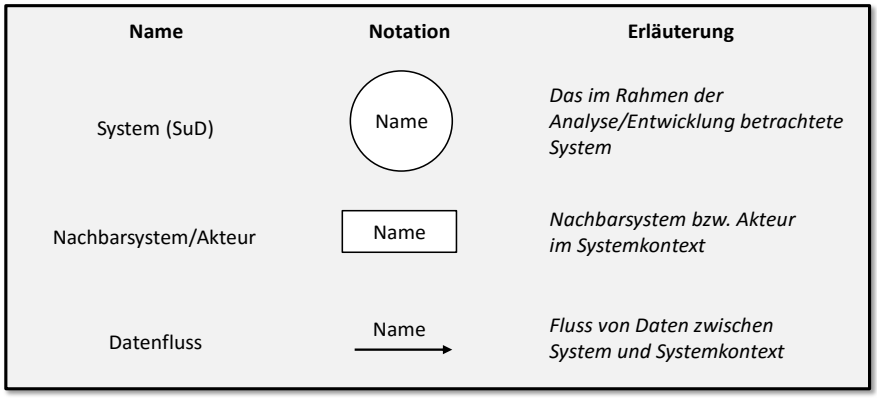

Abbildung 8: Mögliche Modellierungskonstrukte von datenflussorientierten Kontextdiagrammen

Bei der Kontextmodellierung mit Hilfe von Datenflussdiagrammen wird das betrachtete System häufig durch einen Kreis dargestellt, gelegentlich auch als Käschen oder Wolke. Das entsprechende Modellierungskonstrukt repräsentiert das betrachtete System, welches z.B. entweder einen Ausschnitt aus einem Unternehmen, einen Geschäftsprozess oder ein zu automatisierendes System darstellt. Hierdurch wird also der Scope des betrachteten Systems (d.h. die Systemgrenze) ausgedrückt. Die Darstellung der Nachbarsysteme ist relativ beliebig; oft werden diese als Kästchen, aber auch als Strichmännchen oder als 3D-Box bzw. als Doppellinien für externe Datenbanken oder „Files“ modelliert.

In der Strukturierten Analyse nach DeMarco werden die Nachbarsysteme (Quellen und Senken) als Terminatoren (= Endstellen) bezeichnet. Nachbarsysteme bzw. Akteure repräsentieren jegliche Art von Kommunikationsendstellen des betrachteten Systems. Nachbarsysteme bzw. Akteure können einerseits Menschen sein, die mit dem System arbeiten, aber auch beliebige HW/SW-Systeme, Geräte, Sensoren, Aktuatoren, oder passive Datenspeicher (wie Datenbanken oder Dateien), kurzum alles und jeder, der an das System Eingaben liefert oder vom System Ausgaben erhält (oder beides). Die Nachbarsysteme repräsentieren somit Teile des Kontexts des betrachteten Systems.

Die Datenflüsse zwischen Nachbarsystemen bzw. Akteuren und dem betrachteten System repräsentieren Ein- bzw. Ausgabeschnittstellen des zu entwickelnden Systems. Dargestellt weden diese Datenflüsse meist als gerade oder gebogene Linie mit Pfeilspitze zum System (für Eingaben), Pfeilspitze zum Nachbarsystem (für Ausgaben) oder auch als Doppelpfeil. Datenflüsse in einem solchen Kontextdiagramm repräsentieren die ein- und ausgehenden Daten oder Steuerinformationen. Meist werden diese Pfeile als Datenfluss in oder aus dem

System interpretiert. Sollen Steuerflüsse (Kontrollflüsse) bzw. Aufrufabhängigkeiten damit dargestellt werden, so sollte dies in einer Legende zum Diagramm erläutert werden.

2.2.4 Pragmatische Regeln für die Kontextmodellierung mit Datenflussdiagrammen

Als pragmatische Regeln sollte beachtet werden:

- Alle Nachbarsysteme, die mit dem System interagieren, sollten in dem Diagramm enthalten sein (Vollständigkeit der Kommunikationspartner)
- Alle Nachbarsysteme sollten benannt sein (Klarheit, woher die Eingaben kommen und wohin die Ausgaben fließen)
- Alle Ein- und Ausgaben sollten mit logischen Namen der Datenflüsse beschriftet sein (denn unbenannte Pfeile deuten auf ein noch mangelndes Verständnis der Schnittstelle hin)

2.3 Andere Arten der Kontextmodellierung

Die Kooperation zwischen dem betrachteten System und den Nachbarsystemen im Kontext ist auch das Thema der Use-Case-Sicht (vgl. Abschnitt 4.2) und der Szenariosicht (vgl. Kapitel 5). Mit den Use-Cases wird neben der Systemabgrenzung (Scoping) vor allem die Funktionalität des Systems grob gegliedert. Mit der Szenariosicht können zusätzlich zu den Datenflüssen insbesondere Reihenfolgen der Kommunikation und weitere Kommunikationsdetails präziser spezifiziert werden. Es existieren in der Forschung darüber hinaus Vorschläge für die Kontextmodellierung in der zustandsorientierten Sicht, in welcher der Zustandsraum des Systemkontexts und entsprechende Zustandsübergänge modelliert werden, sowie Ansätze, die durch Diagramme der Informationsstruktperspektive statisch-strukturelle Aspekte des Systemkontexts modellieren. Andere Ansätze der Kontextmodellierung betrachten den Systemkontext in der datenflussorientierten Sicht, indem Funktionen im Systemkontext (Kontextfunktionen) modelliert und deren Bezug zu Funktionen des Systems dokumentiert werden. Solche Ansätze dienen insbesondere der maschinellen Aufdeckung ungewollter funktionaler Wechselwirkungen zwischen dem System und seinem Kontext (*Feature Interactions*). Ein Überblick über die verschiedenen Arten der Kontextmodellierung im Requirements Engineering findet sich in [Weye2010].

2.4 Vertiefende Literatur

Datenflussorientierte Kontextdiagramme

- DeMarco, Tom: Structured Analysis and System Specification, Yourdon Press, Prentice Hall, 1979.
- Daun, M.; Tenbergen, B.; Weyer, T.: Requirements Viewpoint. In: Pohl, K.; Hönninger, H.; Achatz, R.; Broy, M.: Model-based Engineering of Embedded Systems, Springer, Heidelberg 2012.

Kontextdiagramme in der Use-Case-Modellierung

- Jacobson, I.; Christerson, M.; Jonsson, P.; Oevergaard, G.: Object Oriented Software Engineering – A Use-Case Driven Approach. Addison-Wesley, Reading, 1992.

3 Informationsstrukturmodellierung

3.1 Zweck

Die Modellierung von Informationsstrukturen nimmt in der Anforderungsmodellierung eine zentrale Rolle ein, da sie im Wesentlichen zwei Aufgaben besitzt:

- Spezifikation von fachlichen Begriffen und fachlichen Daten
- Spezifikation von Anforderungen, die sich auf fachliche Daten beziehen.

Für die Definition von fachlichen Begriffen wird im Requirements Engineering häufig ein Glossar eingesetzt. Hier wird die Bedeutung der Begriffe festgelegt, die zu dem Sprachgebrauch des Fachbereichs (oder allgemeiner: des Auftraggebers) gehören und sich mit dem zu spezifizierenden Problem beschäftigen. Mit der Einführung von Informationsmodellen wird dieser Inhalt eines Glossars um weitere, im Requirements Engineering wichtige Informationen ergänzt. Natürlich tauchen weiterhin die Begriffe aus dem betrachteten Anwendungsfeld auf. Hiermit sind in erster Näherung alle Substantive gemeint, die entweder in textuellen Anforderungen auftreten, oder z.B. bei der daten- oder kontrollflussorientierten Anforderungsmodellierung in der Benennung von Funktionen des Systems (siehe Abschnitt 4.3).

In einem Informationsmodell wird jedoch viel Wert auf die Beziehungen zwischen den Begriffen gelegt. Diese Beziehungen auszudrücken ist eine der Stärken von Diagrammen der Informationsstruktursicht gegenüber einem textuellen, vielleicht alphabetisch geordneten Glossar. Die zweite Erweiterung besteht darin, „Attribute“ zu den Begriffen zu definieren. Attribute drücken aus, welche Eigenschaften bzw. welche fachlichen Daten eines Begriffs relevant sind. So kann in einem Informationsstrukturdiagramm übersichtlich dargestellt werden, welche Eigenschaften z.B. für einen Kunden in einem CRM-System relevant sind. Mit dieser Art von Informationsmodellierung wird also ein konventionelles Glossar um zusätzliche Informationen erweitert. Das Glossar lässt sich aus dieser Art von Diagrammen automatisiert herleiten. Somit wird bei der Verwendung von Informationsmodellen auch der Zweck eines Glossars erfüllt, d.h. die Definition von Begriffen, die einheitlich in der gesamten Systementwicklung genutzt werden sollten.

Eine andere Verwendung für die Modellierung von Informationsstrukturen besteht in der präzisen Spezifikation von Anforderungen. Alle Informationen, die dort festgehalten sind, sollten als Anforderungen betrachtet werden (siehe auch Abschnitt 1.3). Die Aussage von oben, welche Daten eines Kunden für ein CRM-System relevant sind, lässt sich auch als „die Daten, die das CRM-System für einen Kunden verwalten muss“ interpretieren.

3.2 Modellierung von Informationsstrukturen

In diesem Kapitel werden die Anforderungen in der Informationsstruktursicht unter Verwendung von Klassendiagrammen der UML betrachtet. Zur Modellierung von Informationsstrukturen existieren allerdings noch verschiedene andere Ansätze. Ein mit diesem Diagrammtyp verwandtes Diagramm ist das ER-Diagramm [Chen1976]. Heutzutage wird es normalerweise zur Modellierung von Datenbank-Schemata eingesetzt. Die Verwandtschaft zum Klassendiagramm setzt sich in dem Übergang vom Informationsmodell im Requirements Engineering zu einem physikalischen Datenbank-Schema fort. Das Informationsmo-

dell mit seinen fachlichen Daten beschreibt eine gute Grundlage für den Entwurf von Datenbank-Schemata, da dort die fachlichen Inhalte persistiert werden.

Der große Vorteil bei der Verwendung von Klassendiagrammen der UML liegt jedoch in der durch die UML definierte Integration mit anderen Diagrammtypen, die in den anderen Sichten der Anforderungsmodellierung zum Einsatz kommen (vgl. Abschnitt 1.5). Damit können die für ein vollständiges Anforderungsmodell benötigten Verknüpfungen zwischen z.B. Aktivitätsdiagrammen und dem Informationsmodell, formal korrekt und nachvollziehbar erstellt werden.

Aus dieser Integration folgt auch das Vorgehen für die Erstellung eines Informationsmodells im Rahmen des Requirements Engineerings. Üblicherweise wird man mit der Erstellung eines solchen Modells beginnen, um eine gute Grundlage für die Modellierung der weiteren Sichten zu haben. Hierbei wird jedoch schnell ersichtlich, an welchen Stellen noch Defizite in dem Informationsmodell herrschen. Dabei werden dann (vice versa) eventuelle Defizite in Diagrammen anderer Sichten erkannt, da z.B. bei der Definition von Funktionen nicht alle benötigten fachlichen Informationen betrachtet wurden. Dieser Wechsel zwischen den verschiedenen Perspektiven ist nicht immer einfach, birgt jedoch ein großes Potenzial bzgl. der Korrektheit und Vollständigkeit der modellierten Anforderungen.

3.3 Einfaches Beispiel

Abbildung 9 zeigt ein einfaches Beispiel für ein Diagramm der Informationsstruktursicht in Form eines Klassendiagramms der UML, das relevante Begriffe, deren Eigenschaften und die Abhängigkeiten dokumentiert.

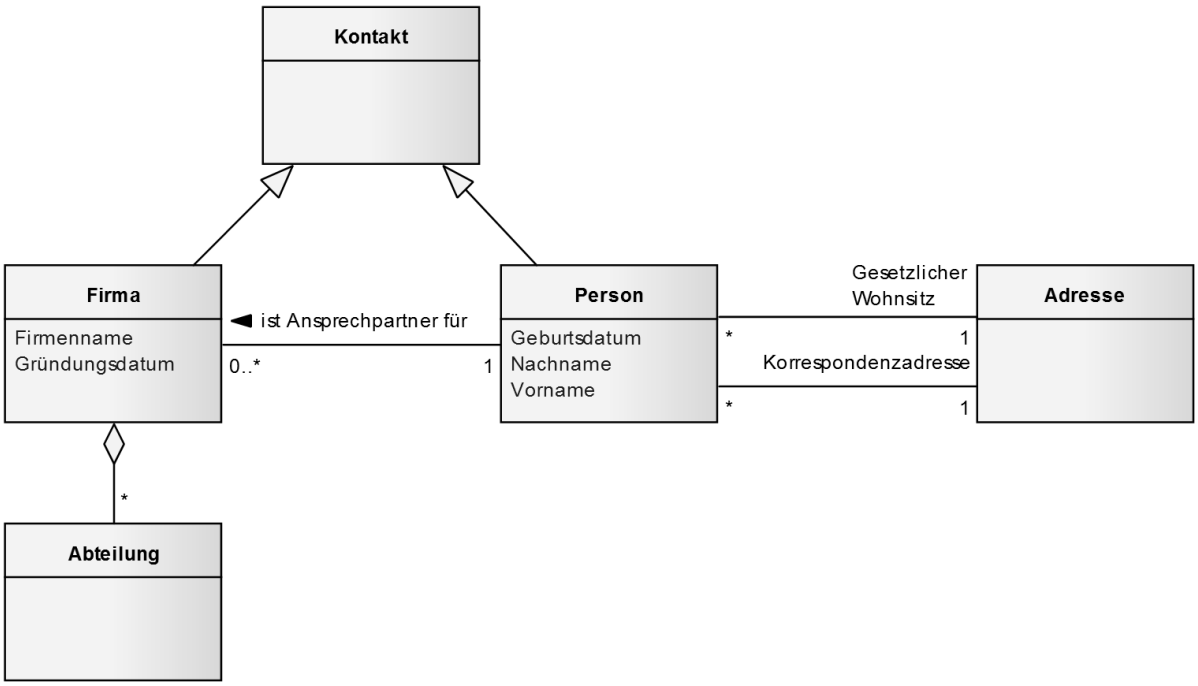

Abbildung 9: Beispiel für ein Klassendiagramm

Das obige Klassendiagramm umfasst die fünf Klassen *Kontakt, Firma*, *Person*, *Adresse* und *Abteilung*. Es dokumentiert wesentliche Eigenschaften dieser Klassen in Form von Attributen, z.B. das Attribut „*Geburtsdatum*“ einer Person, sowie Abhängigkeiten zwischen diesen Klassen, wie z.B. dass eine Person Ansprechpartner für eine Firma ist oder dass eine Firma sich aus Abteilungen zusammensetzt. Die Bedeutung und Verwendung der verschiedenen

Modellierungskonstrukte von Klassendiagrammen in der Anforderungsmodellierung wird in den folgenden Abschnitten im Detail betrachtet.

3.4 Modellierung von Klassen, Attributen und Datentypen

Informationsstrukturdiagramme, die auf Basis von Klassendiagrammen der UML modelliert werden, besitzen als zentrale Elemente die Klasse und deren Attribute.

3.4.1 Klassen

3.4.1.1 Objekte versus Klassen

Bei der Verwendung von Informationsstrukturmodellen in der Anforderungsmodellierung müssen zwei Begriffe voneinander abgegrenzt werden. Man spricht von Objekten und Klassen. Eine „Klasse“ ist als Schablone oder Vorlage zu verstehen, nach der sich die gemeinsamen Eigenschaften vieler Objekte definieren. Die Objekte werden dann als Ausprägungen dieser Klassen bezeichnet.

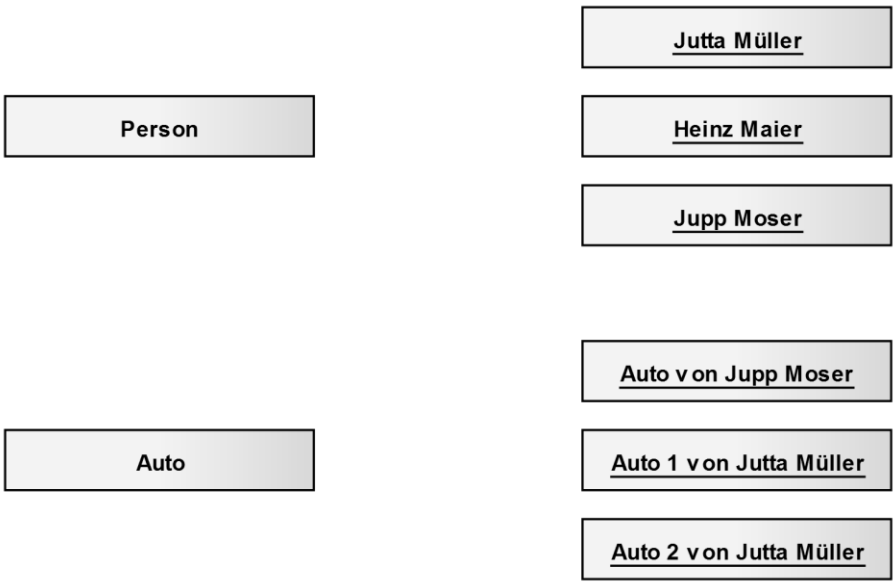

Abbildung 10: Klasse vs. Objekt

In Abbildung 10 sind links die Klassen *Person* und *Auto* und rechts einige Objekte dieser Klassen dargestellt. Bei diesen Objekten ist eine wichtige Eigenschaft der Objekte mit dargestellt: Sie sind eindeutig und sollten demnach auch einen eindeutigen *Identifier* besitzen (mehr zu der Eindeutigkeit findet sich in Abschnitt 3.4.2). Mit dem eindeutigen Namen in der obigen Abbildung lassen sich dann die beiden Autos von Jutta Müller unterscheiden.

3.4.1.2 Syntax und Semantik

Abbildung 11: Eine Klasse

Die einfache Darstellung einer Klasse besteht aus einem Rechteck mit dem Klassennamen. Diese Darstellung wird in Abschnitt 3.4.2 um die Darstellung von Attributen erweitert.

Wie oben erwähnt, repräsentiert eine Klasse die Schablone für eine Vielzahl von Objekten dieser Klasse, die in den Anforderungen referenziert werden. Deshalb wird für den Namen einer Klasse im Allgemeinen der Begriff im Singular verwendet. Wenn von einer Person gesprochen wird, so wäre der Klassenname „*Personen*“ falsch gewählt, da damit eine Menge von Personen gemeint ist.

Die Aussage, dass eine Klasse die Schablone für eine Vielzahl von Objekten dieser Klasse repräsentiert, ist eine für Klassendiagramme allgemein gültige Aussage. Sie lässt sich für den Einsatz der Klassendiagramme zur Modellierung der Informationsstruktursicht eines Anforderungsmodells allerdings präziser formulieren. In den Diagrammen dieser Sicht treten als Klassen die Begriffe auf, die für das betrachtete Anwendungsfeld von Bedeutung sind. Oder anders ausgedrückt: Als Klassen treten die Substantive auf, die zur Formulierung von Anforderungen verwendet werden.

Mit der oben getroffenen Unterscheidung zwischen Objekt und Klasse muss die letztgenannte Aussage präzisiert werden, da in den Anforderungen (textuell oder graphisch) Begriffe verwendet werden, die sich auf ein beliebiges Objekt einer Klasse beziehen.

Beispiel: Das System muss die Daten einer Person anzeigen.

Ausgehend von der Annahme, dass sich in einem Informationsmodell die Klasse *Person* befindet, ist diese Anforderung so zu interpretieren, dass die Daten für jedes Objekt der Klasse *Person* angezeigt werden soll.

Damit ergibt sich die erste Aufgabe der Modellierung des Informationsmodells: Das Identifizieren der benötigten Klassen aus den in den Anforderungen verwendeten Objekten.

3.4.1.3 Heuristiken für die Ermittlung von Klassen

Eine der einfachsten Vorgehensweisen bei der Ermittlung von Klassen ist, für jedes Substantiv aus den Anforderungen (oder den vorliegenden Beschreibungen) eine Klasse zu definieren. Jedoch wird man schnell feststellen, dass dieses Vorgehen eine Unmenge von Klassen liefert, die im Anschluss weiter bearbeitet werden müssen. Viele der so gefundenen Klassen werden „nur“ Eigenschaften einer anderen Klasse beschreiben und werden dann als Attribut dieser Klasse hinzugefügt (siehe Abschnitt 3.4.2). Ein weiterer Punkt bei der Reduktion der vielen Klassen ist es, z.B. Synonyme oder Begriffe außerhalb des Kontextes einzuteilen.

Angenommen, die folgenden Substantive wären in einem ersten Schritt identifiziert worden: Person, Alter, Auto, Geschlecht, Farbe, Fahrzeug, Mensch. In dieser Aufzählung tauchen nur zwei Begriffe auf, die es wert sind (vgl. [Mart1989], [ShMe1988]), als Klassen modelliert zu werden: *Person* und *Fahrzeug*. Für die anderen Begriffe gilt:

- Mensch: Synonym zu Person
- Alter: Eigenschaft einer Person
- Auto: Synonym zu Fahrzeug
- Geschlecht: Eigenschaft einer Person
- Farbe: Eigenschaft eines Fahrzeugs

Bei dieser Festlegung wurden drei Annahmen getroffen, die im Rahmen eines realen Entwicklungsprojektes natürlich bestätigt werden müssen:

- Es soll durchgängig der Begriff *Person* und nicht *Mensch* verwendet werden

- Es soll durchgängig der Begriff *Fahrzeug* und nicht *Auto* verwendet werden
- Der Begriff *Farbe* bezieht sich auf die Farbe eines Fahrzeugs

Für Synonyme ist bei der Wahl des zu verwendenden Begriffs – solange dieser eindeutig ist – der Sprachgebrauch des Projektes oder Unternehmens ausschlaggebend. Mit dieser Vorgehensweise erhält man eine erste gute Version des Informationsmodells. Weiterführende Heuristiken, welche die hier vorgestellte Vorgehensweise erweitern, sind in den Abschnitten 3.4.2.2 und 3.6.3 beschrieben.

Eine andere Möglichkeit, Klassen zu finden, ist, direkt gezielt nach typischen Kandidaten in Formulierungen zu suchen. Diese lassen in drei Bereiche einteilen:

- Materielle oder immaterielle Objekte
- Rollen
- Vorgänge

Mit dieser Aufzählung wird die Menge aller Substantive, wie in dem obigen Vorgehen beschrieben, bereits stark eingeschränkt.

3.4.1.4 Materielle oder immaterielle Objekte

Materielle Objekte der realen Welt sind für die Anforderungen relevant, da sie entweder von dem zu entwickelnden System beeinflusst werden, oder einen „Stellvertreter“ (z.B. eine Klasse) in dem zu entwickelnden Softwaresystem besitzen (oder beides der Fall ist).

Beispiele hierfür sind: *Person*, *Auto*, *Tür*, *Buch*, *Urlaubsantrag* (der nicht ausgedruckt, also nicht materiell sein muss) oder *Verein*.

3.4.1.5 Vorgänge

Für die von dem System unterstützen Prozesse sind häufig weitere Informationen notwendig, die für das System relevant sind, wie z.B. über: Lieferung, Bestellung, Anruf, Montage oder Bericht. Beispielsweise können für eine Lieferung Daten wie das Datum des Eingangs oder der Bearbeiter fachlich relevant für das System sein.

Zu beachten ist hierbei, dass der Begriff im Informationsmodell nicht mit der Funktion gleichgesetzt wird, die das System realisieren soll. Im Informationsmodell werden die für den Prozess relevanten Daten beschrieben – nicht der Prozess selbst, der durch das System unterstützt werden soll (siehe hierzu auch Kapitel 4). Dieser Ablauf wird im Allgemeinen durch ein Substantiv zusammen mit einem Verb in seiner Normalform benannt, nicht wie im Informationsmodell nur durch ein Substantiv.

Eine Bestellung könnte, je nach Anwendungsfeld, eine für das Informationsmodell sinnvolle Klasse sein. Die Entgegennahme einer Bestellung könnte dann eine unterstützende Funktion des Systems sein. Daraus folgen die Namen von z.B. Use-Cases (siehe Abschnitt 4.2): *Bestellung entgegennehmen*, *Bestellung weiterleiten* und *Bestellung abschließen*.

3.4.1.6 Rollen

In gleicher Weise wie bei Vorgängen können auch bzgl. Rollen, die Objekte gegenüber anderen Objekten einnehmen, weitere Informationen von Interesse sein. Diese Rollen werden dann als eigene Klassen definiert. Beispiele hierfür sind:

- *Fahrer*: Eine Person in der Rolle des Fahrers eines Autos
- *Wohnsitz*: Die Adresse des ersten Wohnsitzes einer Person

Für die Modellierung von Rollen steht im Informationsmodell noch eine weitere Alternative zur Verfügung. Mehr Informationen hierzu finden sich in Abschnitt 3.5.1 und Abschnitt 3.7.1.

3.4.1.7 Definition der Bedeutung von Begriffen

Eine wesentliche Eigenschaft eines Informationsmodells ist, dass die dort definierten Begriffe in einen Zusammenhang gesetzt werden (siehe Abschnitt 3.1). Zusammen mit der Definition von Attributen wird im Allgemeinen schon ein großer Teil der Bedeutung eines Begriffes festgelegt[2]. Sollten darüber hinaus noch weitere Beschreibungen notwendig sein, so bietet es sich an, diese in Form von textuellen Ergänzungen zu definieren, die dann mit der entsprechenden Klasse in Beziehung gesetzt werden (siehe Kapitel 0).

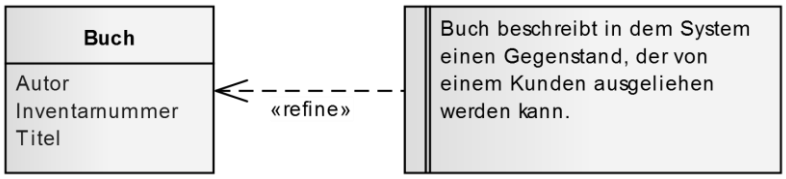

Abbildung 12: Klasse und natürlichsprachliche Definition

3.4.2 Attribute

Klassen werden durch die Verwendung von Attributen genauer spezifziert, d.h. Attributierung reichert die entsprechenden Diagramme mit zusätzlicher Semantik an, was in der Anforderungsmodellierung von großer Bedeutung ist.

3.4.2.1 Syntax und Semantik

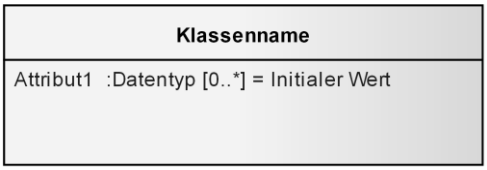

Abbildung 13: Klasse mit Attribut

Die Attribute werden in einem Bereich der Klasse definiert. Dabei sind die folgenden Bestandteile erlaubt (dargestellt in der Backus-Naur-Form):

[/] Name [: Typ] [[Multiplizität]] [= Vorgabewert]

- *Name:* Der Name des Attributs, der angegeben werden muss
- *Datentyp:* Der Datentyp des Attributs. Er ist optional und wird in Abschnitt 3.4.2.4 beschrieben
- *Vorgabewert:* Der Wert des Attributs, der bei Erzeugung als Default gesetzt wird
- *Multiplizität:* Kann verwendet werden, wenn das Attribut mehrere Werte gleichzeitig annehmen kann (Beispiel: Mehrere Vornamen). Hier werden die gleichen Multiplizitäten wie bei Relationen verwendet (siehe Abschnitt 3.5).

[2] Diese Art der Definition unterscheidet sich grundlegend von der eines Glossars in einer textorientierten Spezifikation. Durch das Vorgehen beim Erstellen und die Darstellung ist ein Informationsmodell meist wesentlich fokussierter auf den entsprechenden Kontext zugeschnitten als ein Glossar.

- *Abgeleitet:* Der vorangestellte „/“ deutet an, dass sich der Wert des Attributs aus anderen Werten ableitet (Beispiel: Das Alter lässt sich aus dem Geburtsdatum ableiten).

Die Attribute geben die Eigenschaften einer Klasse an, die fachlich relevant für das zu definierende System sind. Mit anderen Worten: Sie beschreiben die fachlichen Daten, die in dem System genutzt werden müssen.

3.4.2.2 Heuristiken für die Ermittlung von Attributen

Aus der Bedeutung der Attribute für ihre Klassen lässt sich die einfachste Vorgehensweise zur Identifikation von Attributen ableiten. Für jedes Substantiv, das als potenzielle Klasse gefunden wurde (siehe Abschnitt 3.4.1), kann überprüft werden, ob es lediglich eine Eigenschaft eines anderen Begriffes darstellt, der schon als Klasse definiert wurde. Falls ja, so wird dieses Substantiv als Attribut der entsprechenden Klasse definiert.

Häufig werden Attribute auch schon aufgrund von Formulierungen im geschriebenen oder gesprochenen Text als solche erkannt. Häufige Typen von Formulierungen, die auf Attribute von Klassen hinweisen, sind die Folgenden:

3.4.2.2.1 Substantiv in Verbindung mit einem Genitiv

Beispiele: das Bestelldatum der Bestellung; der Durchmesser des Kreises; die Farbe des Autos

Hierbei sind die Namen der Attribute und die zugehörige Klasse schon in den Formulierungen gegeben. Es ist keine weitere Interpretation der Formulierung notwendig.

3.4.2.2.2 Satzkonstruktion mit: <Klasse> hat <Attribut>

Beispiel: eine Person hat ein Geburtsdatum; Eine Adresse hat eine PLZ; Der Prozess hat eine Durchlaufzeit von ...

Eine solche Formulierung deutet auf ein Attribut oder auf eine Beziehung zwischen zwei Klassen hin. Mehr zu der Entscheidung, ob es sich eher um eine Klasse mit einer Beziehung oder um ein Attribut handelt, findet sich in Abschnitt 3.4.2.3.

3.4.2.2.3 Adjektiv in Verbindung mit einem Substantiv

Beispiel: ein schnelles Auto; ein großes Display; ein großes Bankkonto; ein rotes Auto; eine schwarze Liste

Hier ist in der Regel „nur“ eine konkrete Ausprägung des Attributs angegeben (Auto ⇨ schnell). Es muss noch ermittelt werden, welches Attribut durch diese Ausprägung induziert wird. Ist z.B. mit der „schwarzen“ Liste die Farbe des Textes oder die Art der Inhalte dieser Liste gemeint? (siehe Abbildung 14).

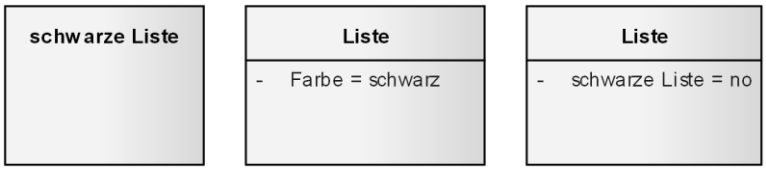

Abbildung 14: Modellierungsvarianten für Adjektive mit Substantiv

3.4.2.2.4 Satzkonstruktionen mit <Klasse> ist <Attributwert>

Beispiel: Falls die Person erwachsen ist, Falls der Antrag genehmigt ist, …

Ähnlich wie in der vorherigen Formulierung ist hier nur ein Wert eines Attributs angegeben. Auch hier ist ein weiterer Schritt notwendig, da in diesem Beispiel eine Klasse mit einem Attributwert verglichen wird. Eigentlich ist jedoch der Vergleich eines Attributs mit dem Wert gemeint.

3.4.2.2.5 Objekte differenzieren

Neben den hier vorgestellten Formulierungen lassen sich Attribute auch aus einer geforderten Eigenschaft von Objekten herleiten.

Aus dem objektorientierten Gedanken folgt, dass Objekte in ihrem Kontext immer eindeutig sein sollten. Diese Eindeutigkeit wird über die aktuelle Belegung der Attribute mit Werten definiert. Zu jedem Zeitpunkt sollte sich die Kombination der Werte zwischen zwei Objekten unterscheiden. Nur dann lassen sich die Objekte von einem Benutzer des Systems auch differenzieren.

Beispiel: Die Modellierung des Objekts „Peter Schulz“ mit lediglich zwei Attributen (Vorname, Nachname) ist eventuell nicht ausreichend, um disese Person von anderen Personen eindeutig zu unterscheiden, die den gleichen Namen haben. Wenn die Klasse „Person“ zusätzlich noch das Attribut „Geburtstag“ besitzt, führt dies dazu, dass die entsprechenden Objekte dieser Klasse eindeutiger voneinander unterschieden werden können (eine andere Person hat ggf. den gleichen Vor- und Nachnamen ist aber mit hoher Wahrscheinlichkeit an einem anderen Tag geboren).

3.4.2.3 Klasse oder Attribut

Die Unterscheidung in Klasse oder Attribut ist nicht immer einfach. Bestehen Zweifel, ob ein identifizierter Begriff im Informationsmodell als Klasse oder Attribut repräsentiert werden soll, sollte der Begriff zunächst als Klasse modelliert werden. Handelt es sich bei dem identifizierten Konzept eher um einfache, unstrukturierte Informationen wie Text, Datum, Zahlen oder boolesche Informationen, sollte der Begriff im Informationsmodell als Attribut repräsentiert werden.

Für strukturierte Informationen gilt: sobald eine Ausprägung dieser strukturierten Information zu mehr als einem anderen Objekt gehört, dann ist für diese strukturierte Information eine Klasse zu modellieren.

Das Beispiel in Abbildung 15 zeigt den Unterschied anhand der Adresse. Objekte der Klasse Adresse können mehreren Objekten der Klasse *Person* zugeordnet sein; diese Objekte teilen sich eine Adresse. Änderungen an einer Adresse wirken sich damit auf alle mit dieser Adresse verbundenen Personen aus. Im Gegensatz dazu sind die Adressen im zweiten Teil des Beispiels völlig unabhängig.

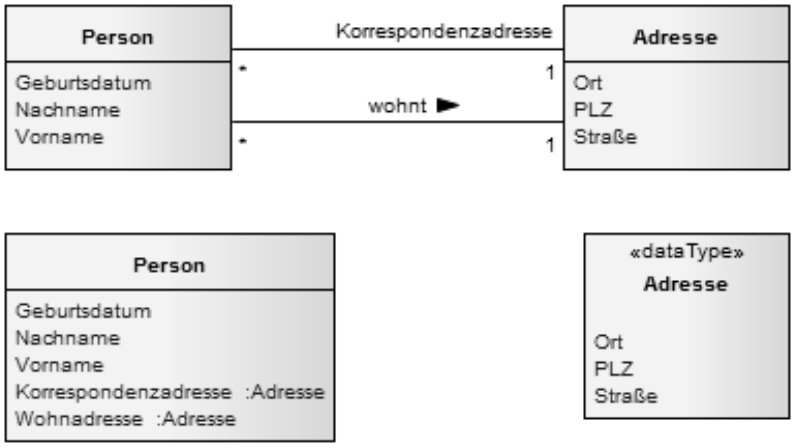

Abbildung 15: Klasse oder Attribut

3.4.2.4 Informationsmodellierung im Umfeld existierender Systeme

Gerade bei existierenden Systemen gibt es einen reichen Fundus an Quellen, die für die Erstellung eines Informationsmodells genutzt werden können. Diese liefern nicht nur mögliche Klassen und Eigenschaften, sondern auch Beziehungen und Multiplizitäten.

Mögliche Quellen:

- Logisches oder technisches Datenmodell (ER-Modelle)
- Schnittstellenbeschreibungen
- Beschreibung eines Data Warehouses

Die Herausforderung bei diesen bestehenden Informationen ist zum einen, wie bei jeder Systemarchäologie, dass die Angaben kritisch hinterfragt und auf Richtigkeit geprüft werden. Andererseits sind meist technische Bestandteile enthalten (technische Identifier und Optimierungen) die in einem Informationsmodell wie es hier betrachtet wird, eher vermieden werden sollten.

3.4.3 Datentypen

Bei der Anforderungsmodellierung mit Klassendiagrammen der UML werden drei unterschiedliche Arten von Datentypen unterschieden, die auch im Rahmen der Anforderungsmodellierung verwendet werden können: *primitive Datentypen*, *strukturierte Datentypen* und *Enumerationen*.

3.4.3.1 Syntax und Semantik

Die Syntax richtet sich nach der Syntax für Klassen. Angegeben wird auf jeden Fall der Name, und, je nach Typ, weitere Informationen. Sie werden verwendet, um die zulässige Wertemenge von Attributen festzulegen.

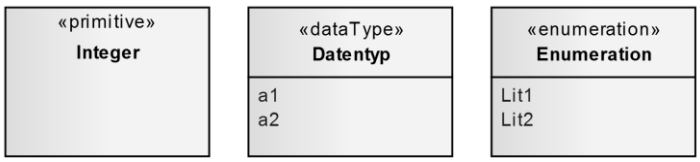

Abbildung 16: Arten von Datentypen

3.4.3.1.1 Primitive – die unstrukturierten Datentypen

Die primitiven Datentypen sind die unstrukturierten und damit einfachsten Datentypen. Sie repräsentieren solche Datentypen wie z.B. eine Zahl, einen Wahrheitswert, eine Zeichenkette etc.

Die UML gibt bereits eine Reihe primitiver Datentypen vor:

- *Boolean*: Ein Wahrheitswert, kann TRUE oder FALSE sein.
- *Integer*: Eine ganze Zahl
- *Float*: Eine Kommazahl
- *Character*: Ein einzelnes Zeichen
- *String*: Eine Zeichenkette

Je nach Anwendungsgebiet kann es sinnvoll sein, weitere primitive Datentypen vorzugeben, also Datentypen zu definieren, die keiner tiefergehenden Definition bedürfen.

Beispiel: String50. Es ist auch ohne weitere Beschreibung eindeutig, dass damit eine Zeichenkette der Länge 50 gemeint ist.

3.4.3.1.2 Datatype – die strukturierten Datentypen

Diese Art von Datentyp[3] erlaubt die Definition von Strukturen, d.h. die Definition komplexerer Datentypen, die sich aus einfacheren Datentypen zusammensetzen. Diese sind immer sehr spezifisch für ein Anwendungsgebiet. Daher wird in der UML lediglich der Mechanismus zur Definition solcher Datentypen vorgegeben und keine konkreten Datentypen. Abbildung 17 zeigt hierzu verschiedene Beispiele.

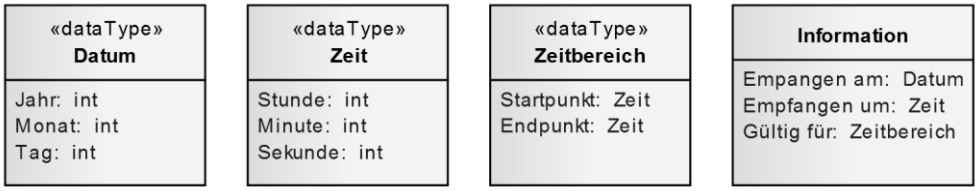

Abbildung 17: Beispiel für die Modellierung und Verwendung von Datatypes

Wie aus dem Beispiel in Abbildung 17 ersichtlich wird, können diese Datentypen hierarchisch definiert werden. Der Ausgangspunkt der hierarchischen Definition sind dabei primitive Datentypen oder Enumerationen.

3.4.3.1.3 Enumerationen – die Aufzählungen

Falls der Wertebereich eines Attributs durch eine Aufzählung der zulässigen Werte angegeben werden kann, so kann dieser Datentyp als eine Enumeration definiert werden Abbildung 18 zeigt zwei Beispiele für die Definition eines Enumeration-Typs.

[3] Leider ist die Bezeichnung der UML für diese Art von Datentyp missverständlich. Gemeint ist hiermit ein Struct oder Record, wie sie aus den älteren, prozeduralen Programmiersprachen bekannt sind.

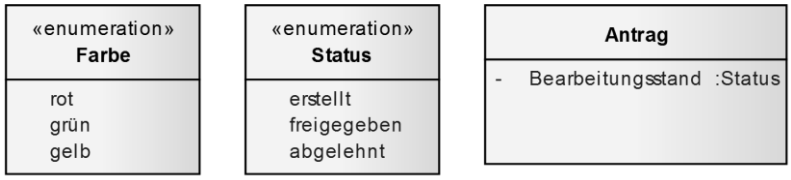

Abbildung 18: Enumerationen

In dem obigen Beispiel ist ein typischer Fall der Verwendung einer solchen Enumeration angegeben: Die Definition eines Status (z.B. eines Antrages). Jedoch ist die Definition dieses Datentyps redundant zu – falls vorhanden – der Definition eines Zustandsautomaten für die entsprechenden Objekte (siehe auch Abschnitt 4.4.4). Deswegen sollte in einem Modell nur eines von beiden enthalten sein.

3.4.3.2 Heuristiken für die Ermittlung von Datentypen

Die erste Frage, die man sich bei der Erstellung eines Informationsmodells im Requirements Engineering stellen muss, ist, ob zu dem jeweiligen Zeitpunkt überhaupt für ein Attribut einer Klasse ein Datentyp angegeben werden muss. Hier ist die Devise, bevorzugt gleich einen Datentyp (wenn auch zunächst einen primitiven Datentyp) zu modellieren. Dieser kann bei Bedarf im Laufe der Modellierung redefiniert bzw. verfeinert werden zu einem komplexeren Datentyp oder einer eigenständigen Klasse. Gegebenenfalls kann der Datentyp auch durch textuelle Anforderungen detaillierter spezifiziert werden.

Die nachfolgende Frage wäre dann, die Bestandteile eines Datentyps zu finden. Für die Enumerationen erschließt sich die Antwort sehr schnell: Man identifiziert die möglichen Werte des Attributs und listet sie in der Enumeration auf. Bei den Strukturen muss man die notwendigen Informationen finden, die einen Wert eines Attributs einer Klasse für das Anwendungsgebiet ausmachen. Dies ähnelt der Fragestellung, wie benötigte Attribute einer Klasse identifiziert werden können (siehe Abschnitt 3.4.2).

3.4.4 Empfehlungen für die Modellierungspraxis

3.4.4.1 Modellierungshinweis „Constraints für Attribute und textuelle Anforderungen“

Reichen die Möglichkeiten der UML für Einschränkungen nicht aus, oder werden Ergebnisse nicht „leicht verständlich“, greift man auf textuelle Anforderungen zurück.

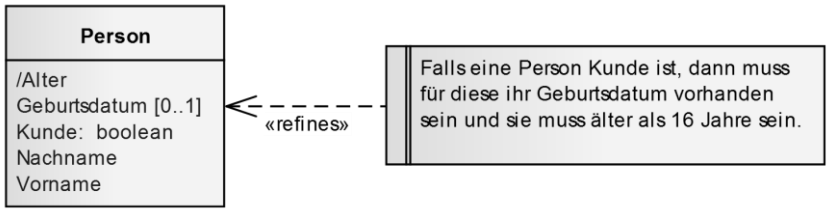

Abbildung 19: Modellierung von Constraints für Attribute

3.4.4.2 Modellierungshinweis „Sichten auf Dinge“

In der Sprache der Projektbeteiligten wird ein Begriff oftmals implizit für mehrere Dinge bzw. Sichten auf ein Ding verwendet (Homonym). So kann z.B. *Antrag* homonym verwendet werden für: *das leere Papierformular*, *das ausgefüllte Dokument* und *das unterschriebene Dokument* oder *die Daten im System*. Aus dem Diagramm muss eindeutig hervorgehen, welche

Bedeutung der oder die modellierten Begriffe haben. Eventuell können Stereotypen dabei helfen, das Modell aussagekräftiger zu machen.

3.4.4.3 Modellierungshinweis „Länge vs. Anzahl von Strings“

Werden für eine Klasse Attribute definiert, die Text enthalten sollen (z.B. Name einer Person), so wird häufig die Frage nach der maximalen Länge gestellt. Bei der Dokumentation dieser Angaben bzgl. des Wertebereichs wird dann häufig – fälschlicherweise – die Angabe einer Multiplizität genutzt. Entsprechend der UML bedeutet jedoch Vorname:String[20] das 20 Vornamen vom Typ String definiert werden– und nicht ein String der Länge 20. Auflösen lässt sich dieses Problem durch die Definition eines speziellen Datatypes.

3.4.4.4 Ausblick „Spezifikation mit OCL“

Für die exakte Definition von Contraints bietet die OCL (Object Contraint Language) der OMG [OMG2012] die Möglichkeit einer formaleren, jedoch nicht einfach verständlichen Spezifikation. Eine Bedingung dass eine Kunde 16 Jahre oder älter sein muss, könnte als OCL Constraint wie folgt formuliert sein: `context Person inv: self.Kunde=true implies self.Alter >= 16`

3.5 Modellierung von Beziehungen

Ein weiterer zentraler Bestandteil eines Informationsmodells sind die Beziehungen. Sie werden als Verbindung zwischen Klassen dargestellt und drücken aus, wie (d.h. mit welcher Bedeutung) die Objekte der jeweiligen Klassen miteinander in Verbindung stehen. Die in der Anforderungsmodellierung gebräuchlichsten Beziehungen sind einfache Beziehungen, Aggregationen und Kompositionen.

3.5.1 Einfache Beziehungen

Einfache Beziehungen[4] werden für Klassen notiert und beschreiben, in welcher Verbindung **zwei** Objekte zueinander stehen. Die zwei Objekte können dabei Instanzen von zwei unterschiedlichen Klassen oder von einer Klasse sein.

Neben den einfachen Beziehungen gibt es in der UML auch die n-ären Beziehungen, die mehrere Objekte verbinden. Diese werden hier jedoch nicht betrachtet.

3.5.1.1 Syntax und Semantik

Assoziationen werden als Linie zwischen den entsprechenden Klassen notiert. Um dieser Linie eine Aussagekraft zu geben, werden weitere Informationen ergänzt. In Abbildung 20 werden die Klassen *Person* und *Adresse* betrachtet. Es soll dokumentiert werden, dass einer Person genau eine Adresse zugewiesen ist, an der sie wohnt und auch genau eine, möglicherweise andere, Adresse, an die die Korrespondenz gesendet werden soll. Eine Adresse kann mehreren Personen als Korrespondenzadresse bzw. Wohnadresse zugewiesen werden.

[4] auch binäre Assoziationen genannt

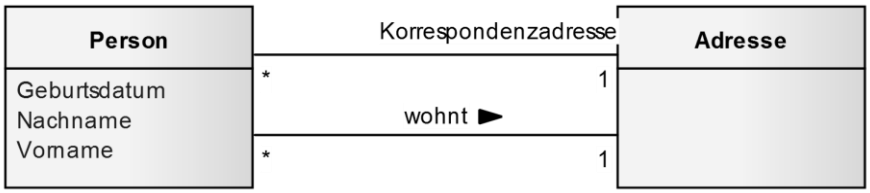

Abbildung 20: Beispiel für die Modellierung einfacher Beziehungen

- *Name:* Gibt den Namen (Bedeutung/Semantik) der Assoziation in Verb-Phrase an
- *Leserichtung:* Richtung, in die der jeweilige Assoziationsname zu lesen ist
- *Multiplizität:* Wird an jedem Ende der Assoziation notiert und gibt an, mit wie vielen Objekten das andere Objekt in Beziehung stehen kann bzw. muss
- *Rolle:* Bezeichnet die Rolle, die das Objekt, an dem die Rolle notiert ist bzgl. des anderen Objekts einnimmt

Für die Angabe dieser Informationen, besonders bei der Multiplizität, ist es hilfreich, sich die Objekte vorzustellen.

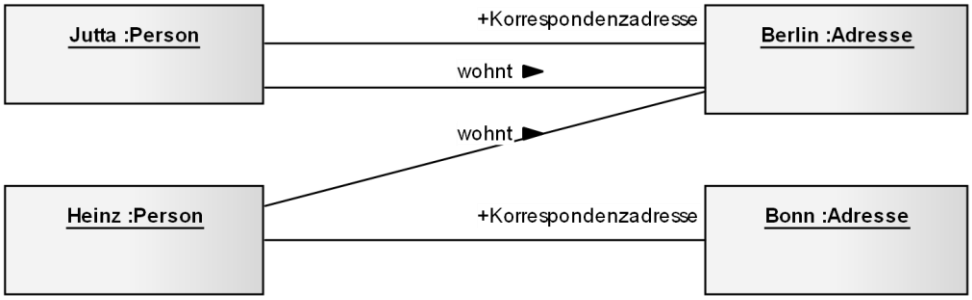

Abbildung 21: Beziehungen der Objekte

Neben den Anforderungen, die im Informationsmodell enthalten sind, lassen sich die Angaben zu den Assoziationen auch für die Formulierung z.B. funktionaler[5] Anforderungen nutzen.

Beispiel: Anforderung ohne Verwendung von Assoziationen

Adresse anzeigen

Eine Funktionalität, wie im Beispiel oben „anzeigen“, die sich nur auf ein Objekt („Adresse“) bezieht, ohne auf dessen Verbindung zu anderen Objekten einzugehen, ist meist lückenhaft. Beziehungen sind dabei die häufigste Form, um genau diesen Kontext zu definieren und damit die Gesamtheit der Objekte auf die gewünschte/geforderte Menge zu reduzieren.

Beispiel: Anforderung mit Verwendung von Assoziationen

Korrespondenzadresse der Person, die Ansprechpartner für die Firma ist, anzeigen

[5] Z.B. Benennung von Aktionen oder Formulierung textueller Anforderungen

Assoziationen bieten die Möglichkeit, sich anhand dieser durch das Informationsmodell zu bewegen. Diese Möglichkeit zur Navigation durch das Informationsmodell zeigt auch, wie wichtig die eindeutige Bezeichnung für die Assoziationen zwischen Klassen ist, gerade wenn mehrere Beziehungen zwischen zwei Klassen existieren. Hierfür muss entweder der Name der Assoziation angegeben oder ein Ende mit einer Rolle bezeichnet sein. Bei der Nutzung von Rollenamen wird bei der Formulierung der Anforderungen der Rollennamen statt der Bezeichnung der Klasse verwendet (siehe Beispiel und Abbildung 9).

Für den Requirements Engineer stellen Multiplizitäten ein wichtiges Mittel dar, um die Angaben der Quantoren in den Anforderungen zu prüfen:

Beispiele:

Anforderung 1: Zeige die Person an

Anforderung 2: Zeige für diese Person die Firma an, für die diese Ansprechpartner ist

Die Formulierung der Anforderung 2 scheint davon auszugehen, dass es genau eine juristische Person gibt. Die Multiplizitäten im Diagramm zeigen ein anderes Bild. Für den Requirements Engineer ergeben sich daher folgende Fragen bzgl. Anforderung und Assoziation:

Stimmt die Multiplizität der Assoziation? Falls diese nicht korrekt sind, müssen sie korrigiert werden; falls diese korrekt sind, dann sind folgende Fragen zu beantworten:

- Was soll passieren, wenn eine juristische Person zugeordnet ist?
- Was soll passieren, wenn mehr als eine juristische Person zugeordnet ist oder was zeichnet die eine aus, die angezeigt werden soll (z.B. die mit dem jüngsten oder ältesten Gründungsdatum)?

3.5.1.2 Heuristik für die Ermittlung einfacher Beziehungen

3.5.1.2.1 Sprachliche Formulierungen

Beziehungen zwischen Klassen lassen sich wiederum durch gewisse Aussagen in der natürlichen Sprache aufdecken. Aussagen wie „Ein Abteilungsleiter <u>leitet</u> eine Abteilung“ können direkt in dem zugehörigen Diagramm ausgedrückt werden. Abhängig von der jeweiligen Formulierung werden solche Aussagen unterschiedlich in einem Klassendiagramm repräsentiert:

Verben → Binäre Assoziation, Assoziationsname, Leserichtung

„Abteilungsleiter <u>leiten</u> Abteilungen“ oder „Abteilungen <u>werden</u> von Abteilungsleitern <u>geleitet</u>“.

Verben in einer aktiv oder passiv Formulierung deuten auf die Bezeichnung der Assoziation hin. Im Modell sollte sie in Aktiv-Form notiert werden. Dienen Anforderungen als Basis für die Ermittlung, muss das Verb (=Funktionalität) kritisch hinterfragt werden.

Beispiel:

Mitarbeiter bestellt Artikel

Im Informationsmodell würde dies als Assoziation nur aufgenommen werden, falls die Information, welcher Mitarbeiter welche Artikel bestellt hat, relevant ist.

Substantive → Rolle

„Mitarbeiter ist <u>Leiter</u> einer Abteilung“

Werden zwei Begriffe mit einem Substantiv in Beziehung gesetzt, handelt es sich meist um eine Rolle, die einer der beiden Begriffe gegenüber dem anderen spielt. Sollte die Rolle Eigenschaften besitzen, dann könnte diese Rolle auch als eigene Klasse modelliert werden (vgl. Abschnitt 3.7.1)

Quantoren → Multiplizität

„eine natürliche Person kann Ansprechpartner für beliebig viele juristische Personen sein“
„für eine justistische Person ist genau eine natürliche Person Ansprechpartner“

Quantoren präzisieren gefundene Assoziationen und sind für beide Enden zwingend notwendig. Die Angabe von „ein/e“ sollte stets mit „genau eins/e“ hinterfragt werden.

3.5.1.2.2 Klassen ohne weiteren Bezug im Klassendiagramm

Jede Klasse im Informationsmodell muss mit mindestens einer anderen Klasse in Beziehung (Generalisierung oder Assoziation inkl. Aggregation oder Komposition) stehen. Sollten Klassen existieren, die mit keiner anderen Klasse verbunden sind, stellt dies eine zu schließende Lücke dar. Präziser bedeutet dies, dass die Klassen und die Beziehungen zwischen ihnen ein Netz bilden.

3.5.2 Aggregation und Komposition

Für eine bestimmte Art der Assoziation (genauer: Semantik von Assoziationen) besitzt UML ein spezifisches Notationselement.

3.5.2.1 Syntax und Semantik

Eine „Teil/Ganzes“-Beziehung kann in der UML mit einer Linie dargestellt werden, an der sich auf der Seite der Klasse, die das Ganze repräsentiert, eine Raute befindet.

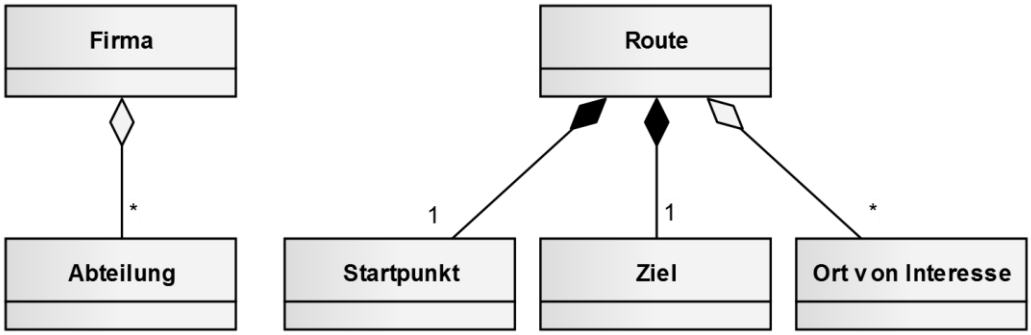

Abbildung 22: Beispiel für die Modellierung von Aggregationen und Kompositionen

Dies ist in erster Linie eine Erleichterung beim Modellieren und Lesen der Diagramme, da auf den ersten Blick die Bedeutung der Assoziation erkennbar ist. Eine besondere Form der Aggregation stellt die Komposition dar. Hier ist die Verbindung Teil/Ganzes besonders stark. Sie wird verwendet um darzustellen, dass beim Löschen oder Vernichten des Ganzen auch die Teile gelöscht werden sollen.

3.5.2.2 Heuristik für die Ermittlung von Aggregationen

Da Aggregationen und Kompositionen als spezifische Arten der Assoziation gelten, können die Heuristiken zur Identifikation von Assoziationen (siehe Abschnitt 3.5.1) auch zur Identifikation von Aggregations- und Kompositionsbeziehungen zwischen Klassen eingesetzt werden. In Anbetracht der spezifischen Bedeutung solcher Assoziationen werden diese durch

Schlüsselwörter angezeigt, die sich auf Aussagen über Teile-Ganzes-Abhängigkeiten beziehen.

Verben

Typische Verben, die Aggregation- bzw. Kompositionsbeziehungen anzeigen sind: besteht aus; setzt sich zusammen aus; beinhaltet; ergibt; hat. Zum Beispiel: *„Eine Firma besteht aus Abteilungen“*.

Substantive

Auch Aggregationen bzw. Kompositionen lassen sich via Rollen-Formulierungen identifizieren. Entsprechend der Bedeutung der Beziehung lauten diese: Teil; Ganzes; Bestandteil; Zum Beispiel *„Eine Abteilung ist Teil einer Firma“.*

3.5.3 Assoziationsklassen

3.5.3.1 Syntax und Sematik

Eine Mischform aus Assoziation und Klassen stellen die sogenannten Assoziationsklassen dar. Sie ermöglichen es, EINE konkrete Assoziation, d.h. die Verbindung zwischen zwei OBJEKTEN, mit Eigenschaften zu belegen.

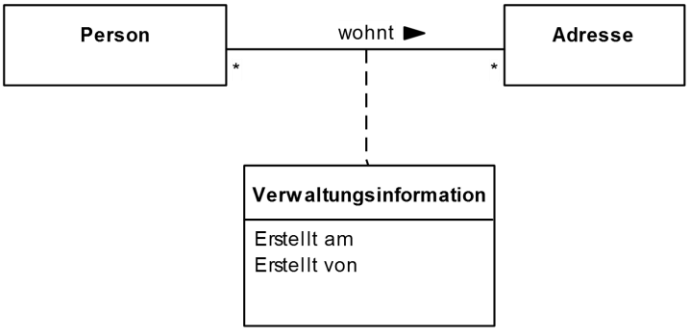

Abbildung 23: Einfaches Beispiel mit Verwaltungsinformationen

In dem obigen Beispiel wird die Verbindung zwischen einem Objekt vom Typ Person und einem Objekt vom Typ Adresse um ein Objekt vom Typ Verwaltungsinformation mit dessen Informationen angereichert, hier also wann und wer diese Verbindung erstellt hat. Zu jeder Verbindung zwischen zwei Objekten besteht also immer ein drittes Objekt, weswegen die Angabe einer Multiplizität zur Assoziationsklasse weder notwendig noch vorgesehen ist.

Eine Modellierung von Assoziationsklassen ist jedoch nicht unumstritten, da sie von ungeübten Lesern oft falsch interpretiert wird. Im Zweifel und ggf. zur Kontrolle des Verständnisses lässt sich eine solche Modellierung auch „klassisch“ darstellen.

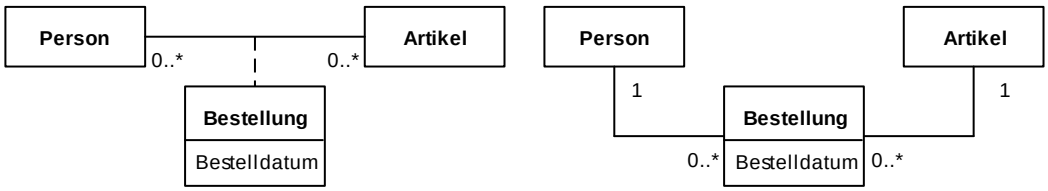

Abbildung 24: Transformierte Modellierung mit „klassischen“ Notationsmitteln

Das Beispiel mit Assoziationsklassen in der Abbildung oben wird auf den ersten Blick gerne - jedoch falsch - interpretiert als: Ein Kunde kann im Rahmen einer Bestellung mehrere Artikel bestellen. Zum besseren Verständnis unten das Beispiel mit Assoziationsklassen aus Abbildung mit Objekten dargestellt.

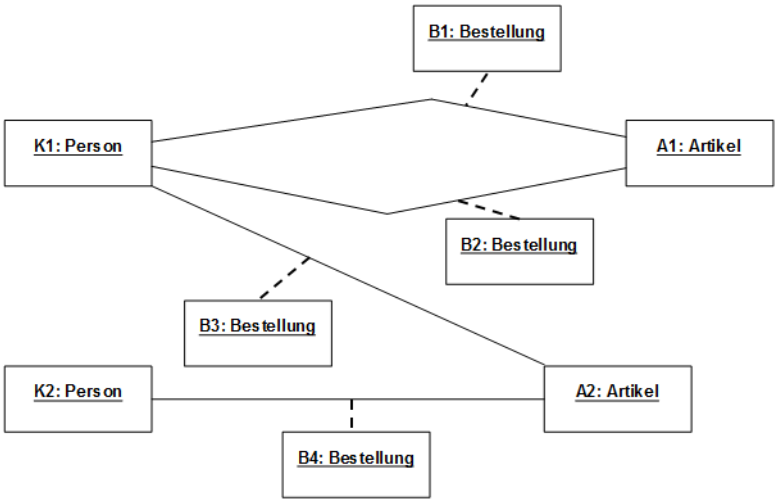

Abbildung 25: Objektdarstellung für das obige Beispiel

Das gezeigte Beispiel kann man so erweitern, dass ein Kunde im Rahmen einer Bestellung auch mehrere Artikel bestellen kann, z.B. durch ein Attribut Menge in der Klasse Bestellung.

3.5.3.2 Heuristik für die Ermittlung von Assoziationsklassen

Assoziationsklassen attributieren Assoziationen. Sprachlich sind damit alle Formulierungen interessant, die sich auf Eigenschaften bzgl. einer Assoziation beziehen.

Beispiel:

Wie lange / seit wann eine Person an einer Adresse wohnt.

Wann / wie häufig eine Person einen Ort besucht hat.

3.5.4 Empfehlungen für die Modellierungspraxis

3.5.4.1 Modellierungshinweis „Constraints von Beziehungen und textuelle Anforderungen“

Reichen die Möglichkeiten der UML für Einschränkungen nicht aus, oder werden Ergebnisse nicht „leicht verständlich“, greift man auf textuelle Anforderungen zurück.

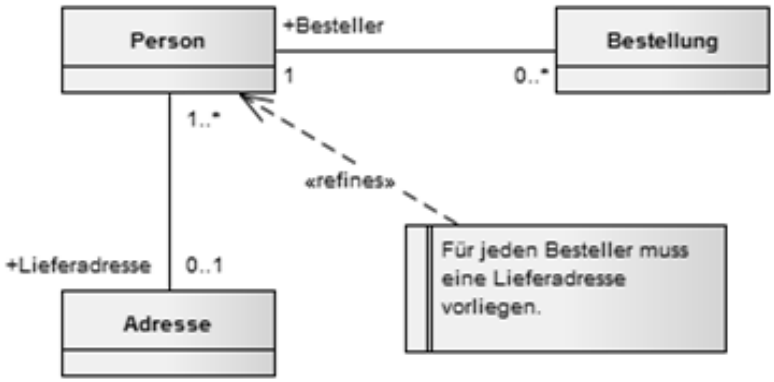

Abbildung 26: Modellierung von Constrains von Beziehungen

3.5.4.2 Modellierungshinweis „Attribut oder Assoziation“

Zwei Klassen, die mit einer 1:1 oder 1:0..1 Beziehung miteinander verbunden sind, können vorkommen, sind jedoch eher ungewöhnlich. Hier sollte hinterfragt werden, ob eine der beiden Klassen in ein Attribut überführt werden kann/müsste.

3.5.4.3 Modellierungshinweis „Navigierbarkeit vs. Leserichtung“

Bei der Modellierung von Klassen gibt es zwei Darstellungen von Beziehungen, die als „Pfeile“ interpretiert werden können (den Generalisierungs-„Pfeil“ nicht mitgerechnet), jedoch eine andere Bedeutung haben. Die eine ist die Leserichtung der Bezeichnung der Assoziation (vgl. Abschnitt 3.5.1).

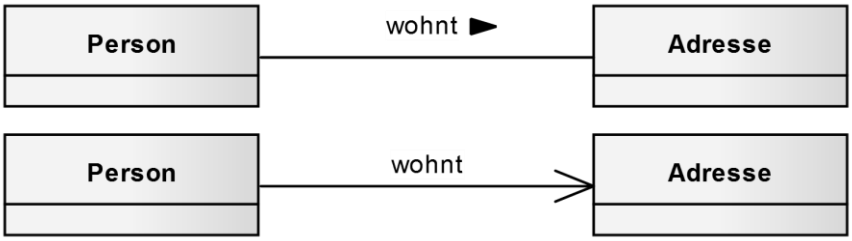

Abbildung 27: Leserichtung vs Navigierbarkeit

Eine andere ist die Navigierbarkeit (siehe Abbildung 27). Letztere sagt aus, dass eine Person die Adresse kennt, an der sie wohnt, nicht jedoch umgekehrt. Diese Navigierbarkeit ist in der Realisierung wichtig, spielt im Requirements Engineering jedoch eine untergeordnete Rolle.

3.5.4.4 Modellierungshinweis „Unterschiedliche Interpretation von Multiplizitäten (Versionierung, Historisierung, Dynamik)“

Multiplizitäten scheinen sehr exakt definiert zu sein, führen bei Diskussionen allerdings dennoch teilweise zu unterschiedlichen Interpretationen.

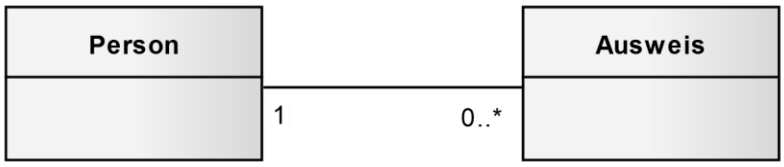

Abbildung 28: Unklare Multiplizitäten

0..* interpretierbar als:

- *: Person hat (im Laufe der Zeit) viele Ausweise (abgelaufene, verlorene)
- 0: Person benötigt keinen Ausweis (hat noch keinen oder hat ihn verloren)
- 0: Eine Person hat immer einen Ausweis, aber zuerst wird eine Person angelegt, dann der Ausweis. Also gibt es einen kurzen Zeitraum, ehe die Daten gespeichert werden, in dem eine Person ohne Ausweis existiert.

Ein Informationsmodell zeigt immer eine statische und konsistente Struktur der Informationen. Entsprechend ist es nicht dafür gedacht, Strukturen im „Zwischenspeicher“ zu berücksichtigen. Andere zeitliche Aspekte, z.B. Versionierung, einfache oder zweidimensionale Historisierung, können durchaus relevant sein und entsprechend modelliert werden. Abbildung 29 zeigt eine mögliche Modellierung einer einfachen Historie.

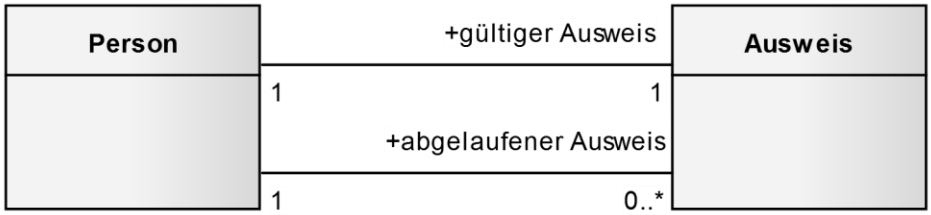

Abbildung 29: Auflösung unklarer Multiplizitäten

3.5.4.5 Ausblick „Spezifikation mit OCL“

Für die exakte Definition von Contraints bietet die OCL [OMG2012] die Möglichkeit einer formaleren, jedoch nicht einfach verständlichen Spezifikation. Die Bedingung, dass jede Person in der Rolle *Besteller* auch eine Lieferadresse besitzen muss, könnte z.B. durch den folgenden OCL Constraint ausgedrückt sein:

```
context Bestellung
inv:self.Besteller->notEmpty()implies
self.Besteller.Lieferadresse->notEmpty()
```

3.6 Modellierung von Generalisierungen und Spezialisierungen

3.6.1 Syntax und Semantik

Gleichen sich zwei oder mehrere Klassen, können die gemeinsamen Eigenschaften und Beziehungen durch eine Generalisierung zusammengefasst und das Modell somit vereinfacht werden. Die entsprechenden Klassen werden durch eine Linie mit einem Dreieck an einem Ende verbunden, wobei die Klasse, die mit dem Dreieck verbunden ist, den generalisierten Begriff darstellt. Für den Fall, dass es von dem generalisierten Begriff keine eigenen Objekte (Instanzen) geben wird, spricht man von einer abstrakten Klasse. Um dies im Diagramm darzustellen, wird der Name dieser Klasse kursiv dargestellt. Abbildung 30 zeigt ein einfaches Beispiel für die Modellierung einer Generalisierungsbeziehung.

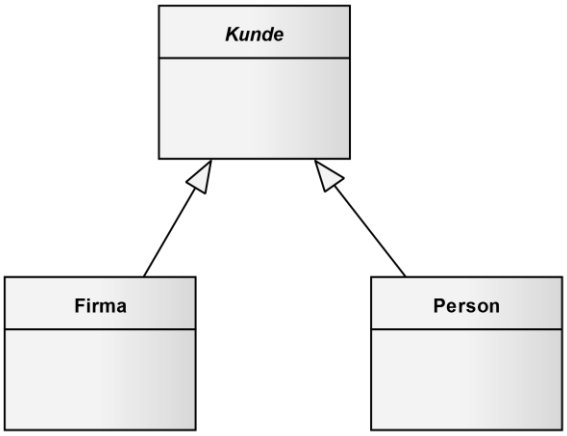

Abbildung 30: Beipiel für die Modellierung von Generalisierungen

Generalisierte Begriffe sollten mit Bedacht in Anforderungen verwendet werden, da hier die Gefahr von Missverständnissen besteht. Abstrakte und nicht abstrakte Generalisierungen haben für Anforderungen eine unterschiedliche Bedeutung. Abstrakte Generalisierungen stehen in Anforderungen, im Gegensatz zu nicht abstrakten Generalisierungen, stellvertretend für jede ihre Spezialisierungen.

Das System muss dem Anwender die Möglichkeit bieten, *Kunden*<abstrakte Generalisierung> anzulegen.

Entspricht: 1) Das System muss dem Anwender die Möglichkeit bieten, *Firmen*<Spezialisierung1> anzulegen; 2) Das System muss dem Anwender die Möglichkeit bieten *Personen*<Spezialisierung2> anzulegen.

Im Gegensatz dazu bezieht sich eine Anforderung mit einer <nicht abstrakten Generalisierung > eben nur genau auf diese.

3.6.2 Generalisierungsmengen und deren Constraints

Generalisierungsmengen bieten die Möglichkeit, verschiedene Aspekte einer Generalisierung zu Gruppen von Subtypen zusammenzufassen. In Abbildung 31 sind die Generalisierungsmengen *Kontaktart* und *Kontakttyp* mit zugehörigen Constraints modelliert.

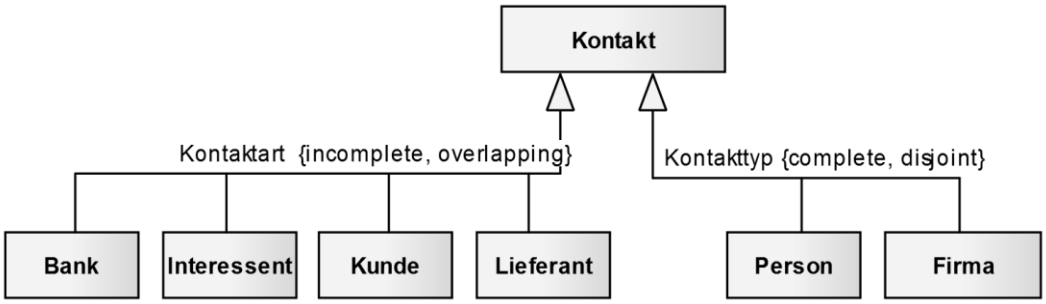

Abbildung 31: Beispiel für die Modellierung von Generalisierungsmengen und Contraints

Die Spezifikation von Eigenschaften einer solchen Gruppe ist in der UML vorgegeben und wird durch Constraints in geschweiften Klammern annotiert. Typische Constraints für Generalisierungsmengen sind:

- *Incomplete:* Die modellierten Subtypen sind nicht zwingend vollständig. Zum Beispiel könnten weitere Kontaktarten wie Hersteller etc. hinzukommen.

- *Complete:* Die modellierten Subtypen sind vollständig. Zum Beispiel gibt es keine anderen Kontakttypen.
- *Disjoint:* Eine Instanz kann nur Ausprägung von einem Subtyp sein. Zum Beispiel ist ein Kontakt entweder eine Person oder eine Firma.
- *Overlapping:* Eine Instanz kann Ausprägung von mehr als einem Subtyp sein. Zum Beispiel kann ein Kontakt Kunde und auch Lieferant sein.

3.6.3 Heuristik für die Ermittlung von Generalisierungen

3.6.3.1 Sprachliche Formulierung

Wie in den anderen Bereichen lassen sich Generalisierungen bzw. Spezialisierungen auch durch spezifische sprachliche Formulierungen finden.

„Der Hund ist eine Art von Tier"; „Eine Art von Tier ist der Hund"; „ein spezieller Mitarbeiter ist der Chef"; „Typische Zahlungsmethoden sind Überweisung oder Rechnung"

3.6.3.2 Gleichartigkeit

Für Klassen, die sehr viele gleiche Attribute besitzen und evtl. auch gleiche Beziehungen zu anderen Klassen haben, können künstliche generalisierte Klassen erstellt werden. Hierdurch werden unter Umständen künstliche abstrakte Begriffe geschaffen, für die es keine fachliche Notwendigkeit gibt.

3.6.4 Empfehlungen für die Modellierungspraxis

Sollten nur leere Spezialisierungen vorkommen, wäre auch die Modellierung via einer Eigenschaft „Typ" oder „Art" möglich.

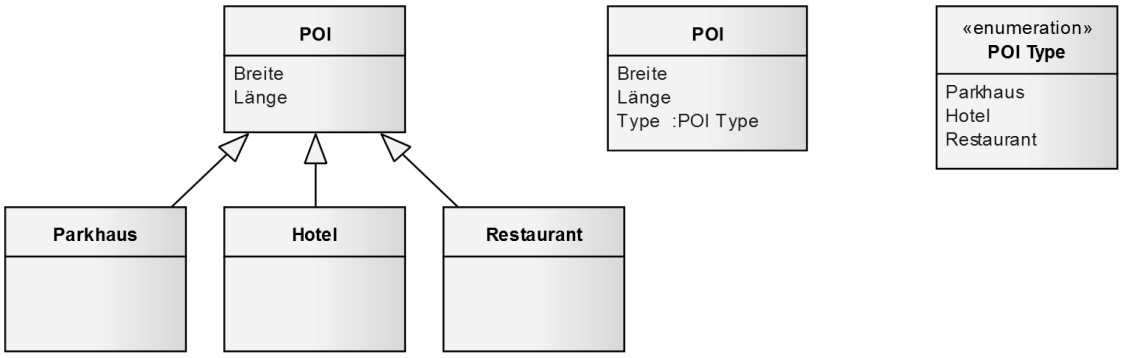

Abbildung 32: Leere Spezialisierungen

Über die richtige Modellierung entscheidet hier der Sprachgebrauch. Sind die Spezialisierungen als eigenständige Begriffe in der Sprache der Stakeholder stark verankert, so sollten diese auch als eigenständige Begriffe modelliert werden. Spielen sie eine untergeordnete Rolle und werden eher mit „<Begriff> vom Typ <x>" verwendet, ist die Modellierung als Typ vorzuziehen.

3.7 Weiterführende Modellierungskonzepte

3.7.1 Typische Konzepte und Muster für Informationsstrukturmodellierung

Bei der Modellierung von Informationsmodellen trifft man immer wieder auf gleichartige Strukturen in den Modellen. Mögliche Lösungen solcher Strukturen werden als Pattern oder Muster bezeichnet. Die wichtigsten Analysemuster für Informationsmodelle sind:

- Item-Item Description, z.B. für Titel und Exemplar; Produkt und Artikel; Rechnung und Rechnungsposition [CoNM1996]
- Party (auch bekannt als Rollen-Pattern) [Fowl1996]
- Koordinator für z.B. Prozesse [Balz2011]
- Kompositum (Composite) z.B. für Organisation oder Dateisystem [GaJV1996]

3.7.2 Abgeleitete Assoziation

Abgeleitete Assoziationen sind Assoziationen, die sich aus vorhandenen Informationen herleiten lassen – und damit redundant sind. Ähnlich wie bei abgeleiteten Attributen benötigen diese Assoziationen eine Ableitungsvorschrift, die im einfachsten Fall textuell ergänzt wird und das Formulieren von Anforderungen sehr vereinfachen kann, da die Ableitungsvorschrift nur einmal definiert werden muss. Abbildung 30 zeigt die Modellierung einer abgeleiteten Assoziation.

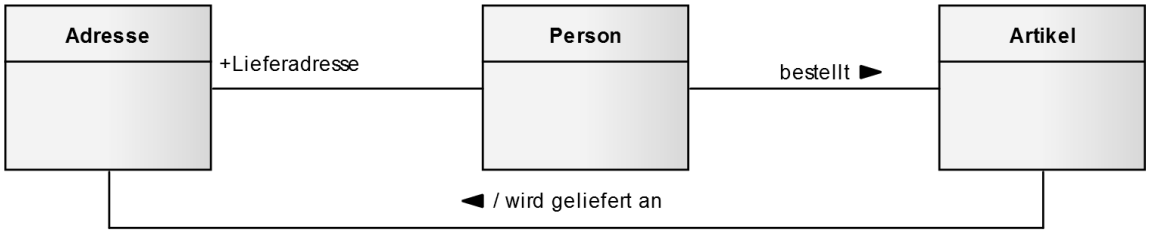

Abbildung 33: Abgeleitete Assoziationen

3.7.3 Umfang von Generalisierungsdiagrammen

Generalisierungen können schnell ganze Bäume mit mehreren Ebenen bilden. Sobald ein solcher Baum aus mehr als 7±2 Elementen besteht, sollte dieser in einem eigenen Diagramm festgehalten werden.

3.8 Vertiefende Literatur

Erstellen von Informationsmodellen

- Martin, J.: Information Engineering, Book I – Introduction. Prentice Hall, Englewood Cliffs, 1989.
- Shlaer, S.; Mellor, S.: Object-oriented Systems Analysis – Modeling the World in Data. Prentice Hall, Englewood Cliffs 1988.
- Rupp, C.; Queins, S.: UML 2 Glasklar – Praxiswissen für die UML-Modellierung. Carl Hanser Verlag 2012.

- DeMarco, T.: Structured Analysis and System Specification, Yourdon Press, Prentice Hall, 1979.
- Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, Reading, MA 2004.

Analyse-Pattern für Informationsmodelle

- Coad, P.; North, D.; Mayfield, M.: Object Models: Strategies, Patterns, and Applications, Prentice Hall, 1996.
- Fowler, F.: Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading, MA 1996.
- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Entwurfsmuster. 5 Auflage. Addison-Wesley, 1996.
- Rupp, C.; Queins, S.: UML 2 Glasklar – Praxiswissen für die UML-Modellierung. Carl Hanser Verlag 2012.

4 Dynamische Sichten

Programm = Daten + Algorithmen! Mit dieser markanten Feststellung hat Nikolaus Wirth einen komplexen Sachverhalt einprägsam zusammengefasst. Überträgt man diese Gleichung auf die Anforderungen, so stehen in diesem Teil (nach der Informationsstrukturmodellierung in Kapitel 3) die gewünschte oder geforderte Funktionalität eines Systems und deren Verhalten im Mittelpunkt.

4.1 Dynamische Sichten der Anforderungsmodellierung

Im Gegensatz zu den Informationsmodellen, die (bis auf syntaktische Varianten) im wesentlichen durch einen Diagrammtyp ausgedrückt werden können, bieten die dynamischen Sichten eine Menge an unterschiedlichen Abstraktionskriterien, mittels derer unterschiedliche Aspekte der Funktionalität in den Vordergrund gerückt werden können (siehe Kapitel 0). Dieses Kapitel betrachtet vier Arten dynamischer Sichten in der Anforderungsmodellierung, die in der folgenden Tabelle zusammengefasst sind

Sicht	Bedeutung
Use-Case-Sicht	Zerlegung der Funktionalität des Gesamtsystems aus einer Nutzungssicht in extern (oder zeitlich) ausgelöste Prozesse (bzw. Interaktionen bzw. Abfolgen von Funktionen), die jeweils zu einem spezifischen fachlichen Mehrwert für einen oder mehrere Akteure im Systemkontext führen, dargestellt in Form von Use-Case-Diagrammen mit jeweils die Use-Cases detaillierenden Use-Case-Spezifikationen
Kontrollflussorientierte Sicht	Spezifikation von Folgen geforderter Funktionen eines Systems, wobei der Schwerpunkt auf der Reihenfolge der Ausführung liegt. Dargestellt wird diese Sicht hauptsächlich durch UML-Aktivitätsdiagramme mit erläuternden Aktivitätsbeschreibungen.
Datenflussorientierte Sicht	Spezifikation von geforderten Funktionen eines Systems mit ihren Ein-/Ausgabeabhängigkeiten, dargestellt klassischerweise durch Datenflussdiagramme mit erläuternden Beschreibungen der Funktionen und der Datenflüsse zwischen den Funktionen, aber auch durch entsprechende Erweiterungen der UML-Aktivitätsdiagramme.
Zustandsorientierte Sicht	Spezifikation der Funktionen eines Systems in Abhängigkeit von Zuständen, in denen diese Funktionen ausgeführt werden dürfen. Dargestellt in Form von Zustandsübergangsdiagrammen oder Statecharts mit erläuternden Beschreibungen der Funktionen und der Ereignisse, die Zustandswechsel auslösen.
Szenariosicht (Kapitel 5)	Spezifikation von Interaktion zwischen Akteuren (Personen, Systeme) im Systemkontext und dem SuD, die zu einem fachlichen Mehrwert für einen oder mehrere Akteure im Systemkontext führen. Die Anforderungsmodellierung kann exemplarsich angewendet werden (z.B. zur Unterstützung der Gewinnung von Anforderungen) oder mit dem Anspruch auf Vollständigkeit, d.h. sämtliche Szenarien, die vom SuD erfüllt werden sollen.

Tabelle 1: Dynamische Sichten in der Anforderungsmodellierung und deren Bedeutung

4.2 Use-Case-Modellierung

Use-Cases (Anwendungsfälle) stellen eine Möglichkeit dar, Funktionen innerhalb des festgelegten Scopes systematisch aus Benutzersicht zu zerlegen. Neben der Betrachtung der Grun-

delemente von Use-Case-Modellen steht ein tieferes Verständnis darüber, wie man Use-Cases identifiziert und spezifiziert, im Mittelpunkt dieses Abschnitts.

4.2.1 Zweck

Um die Funktionalität eines komplexen Gesamtsystems in Teile zu zerlegen, stehen vielfältige Ansätze zur Verfügung. Umfassend bewährt hat sich eine Aufgliederung des Gesamtsystems in Prozesse (vgl. [McPa1984],[JCJO1992], [HaCa1993], [Cohn2002]), die für Personen oder Systeme außerhalb des betrachteten Systems einen Nutzen (d.h. einen Mehrwert) erbringen. Für solche Prozesse werden unterschiedlichste Konzepte und Bezeichnungen verwendet, von EPK (Ereignis-Prozess-Kette) über Use-Case bis hin zu User-Story in agilen Vorgehensweisen.

Wir betrachten Use-Case-Modelle als einen Vertreter dieser Modelle. Use-Case-Modelle bestehen aus Use-Case Diagrammen mit dazugehörigen Use-Case-Spezifikationen. Die Use-Case-Diagramme geben einen grafischen Überblick über die geforderten Prozesse des Systems und ihre Beziehungen zu Akteuren in der Systemumgebung. Eine Use-Case-Spezifikation spezifiziert jeweils einen Use-Case im Detail, in dem z.B. mögliche Abläufe des Use-Cases sowie Vor- und Nachbedingungen der Ausführung des Use-Cases dokumentiert sind. Die Spezifikation von Use-Cases erfolgt im Wesentlichen textuell, z.B. mit Hilfe von Use-Case-Schablonen wie sie z.B. in [Cock2000] vorgeschlagen werden.

Der Zweck von Use-Case-Modellen besteht hauptsächlich darin, ein komplexes System nach dem Motto „Teile und beherrsche“ in solche Teile zu zerlegen, die anschließend möglichst unabhängig voneinander detaillierter spezifiziert werden können. Da die Prozesse (= Use-Cases) aus dem Kontext ausgelöst werden, ist diese Zerlegung neutral gegenüber der (bisherigen oder geplanten) inneren Struktur des Systems, d.h. sie berücksichtigt keinerlei innere Organisationsgrenzen, Software- oder Hardwaregrenzen des betrachteten Systems, sondern konzentriert sich auf die Außensicht.

4.2.2 Notationselemente für Use-Case-Diagramme

Abbildung 34 zeigt die wesentlichen Modellierungskonstrukte von Use-Case-Diagrammen, wie sie z.B. auch in der UML Verwendung finden und zwar zur Modellierung der Sytemgrenze, von Akteuren, Use-Cases und den Beziehungen zwischen Akteuren und Use-Cases. Zum Begriff des Akteurs ist anzumerken, dass Akteure immer auch Stakeholder im Sinne des Requirements Engineeings sind, aber viele Stakeholder keine Akteure, da sie nie mit dem System im Betrieb zusammenarbeiten, auch wenn sie das Verhalten des Systems mitbestimmen wollen (vgl. [Cock2000]).

Für Akteure können (neben dem Strichmännchen) auch andere grafische Stereotyps verwendet werden. Unter anderem hat sich die Verwendung eines Uhr-Symbols für zeitgetriggerte Prozesse bewährt, wie es auch in Abb. 32. verwendet wird.

Anmerkung: Da in Use-Case-Diagrammen durch die Vorgabe der Systemgrenze eindeutig „innen“ und „außen“ zu unterscheiden ist, sind Akteure in jeglicher Art der Darstellung auch ohne dem Stereotyp <<Actor>> gut zu erkennen. Viele Modellierungstools erlauben das Anzeigen oder Ausblenden von Stereotypbezeichnungen wie <<Actor>>. Abbildung 33 macht von dieser vereinfachten Notation Gebrauch.

Notation	Name	Bedeutung
Name	Systemgrenzen	Das Rechteck stellt den Scope des Systems dar. Akteure sind außerhalb des Scopes. Use-Cases sind innerhalb des Scopes.
Name (alternativ) «actor» Name	Akteur	Ein Akteur kann eine Person, eine Firma oder Organisation, eine Software oder ein Systemelement (Hardware, Software oder beides) sein.
Name	Use Case	Eine von einem Akteur benötigte Funktionalität des Systems, die dem Akteur einen Mehrwert erbringt. Der Name sollte eine Verbform enthalten, da es sich um eine Funktionalität handelt und ein Objekt, auf das sich diese Funktionalität bezieht, z.B. „Geschwindigkeit überwachen“. Teils finden sich Namen auch in substantivierter Form, d.h. z.B. Geschwindigkeitsüberwachung“).
Name	Assoziation	Die (unbenannte) Linie zwischen Akteur und Use-Case drückt aus, dass dieser Akteur mit jenem Use-Case interagiert.

Abbildung 34: Modellierungskonstrukte von Use-Case-Diagrammen

Abbildung 35 zeigt im rechten Bereich ein Beispiel für ein Use-Case-Diagramm mit diesen vier Grundelementen, d.h. Systemgrenze (Scope), Akteur, Use-Case und Assoziation.

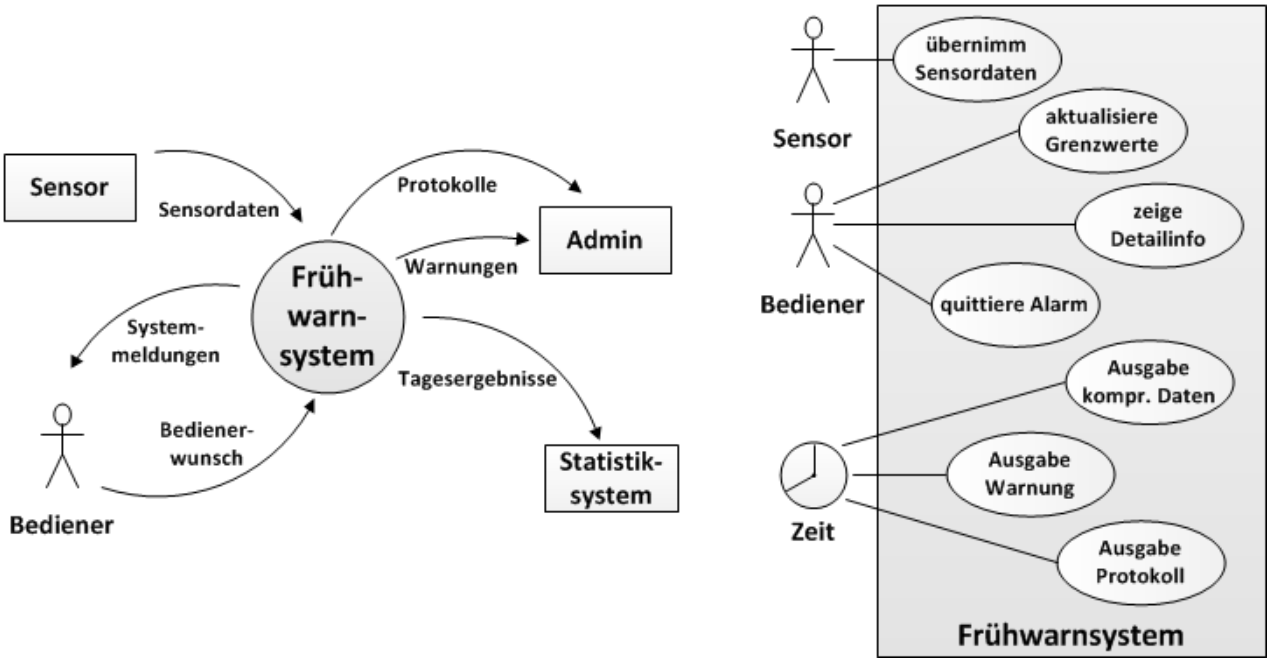

Abbildung 35: Bsp. für Kontext-Diagramm (links) und für korrespondierendes Use-Case-Diagramm (rechts)

4.2.3 Use-Case-Diagramme und Kontextdiagramme

Diese beiden Diagrammarten haben ähnliche Inhalte und doch unterschiedliche Schwerpunkte. Beide legen einen Namen für das betrachtete System fest und definieren die Systemgrenze (d.h. die Abgrenzung zwischen Scope und Kontext), jedoch mit unterschiedlicher Präzision.

Der Schwerpunkt des Kontextdiagramms liegt in der präzisen, fachlichen Festlegung der Schnittstellen zu *allen* Nachbarsystemen. Gute Kontextdiagramme enthalten (neben dem

System als Black Box) *alle* Nachbarsysteme (Menschen, IT-Systems, Geräte), von denen Informationen für das betrachtete System kommen oder an die Informationen aus dem System fließen.

Existiert ein Kontextdiagramm, in dem sämtliche Akteure (d.h. z.B. andere Systeme, Personen/Rollen) des betrachteten Systems aufgeführt sind, so kann ein Use-Case-Diagramm auch derart verwendet weden, dass lediglich die prozessauslösenden Akteure, d.h. diejenigen Akteure, die die Ausführung eines Use-Cases auslösen, modelliert werden. Durch diese „Akteure“ wird die Existenz von Use-Cases begründet. Mit anderen Worten, ohne den jeweiligen Akteur gäbe es keine Forderung nach diesem Use-Case. Liegt also ein Kontextdiagramm vor, so müssen andere Akteure, die an dem Use-Case beteiligt sind (d.h. die im Laufe der Abarbeitung des Prozesses nach Start durch einen Akteur) nicht notwendigerweise eingezeichnet werden. Sie vergrößern nur die Komplexität des Use-Case-Diagramms, und lenken davon ab, dass die Use-Case-Sicht hauptsächlich dazu dient, die Gesamtfunktionalität eines Systems aus der Nutzungssicht in möglichst disjunkte Teilprozesse zu zerlegen.

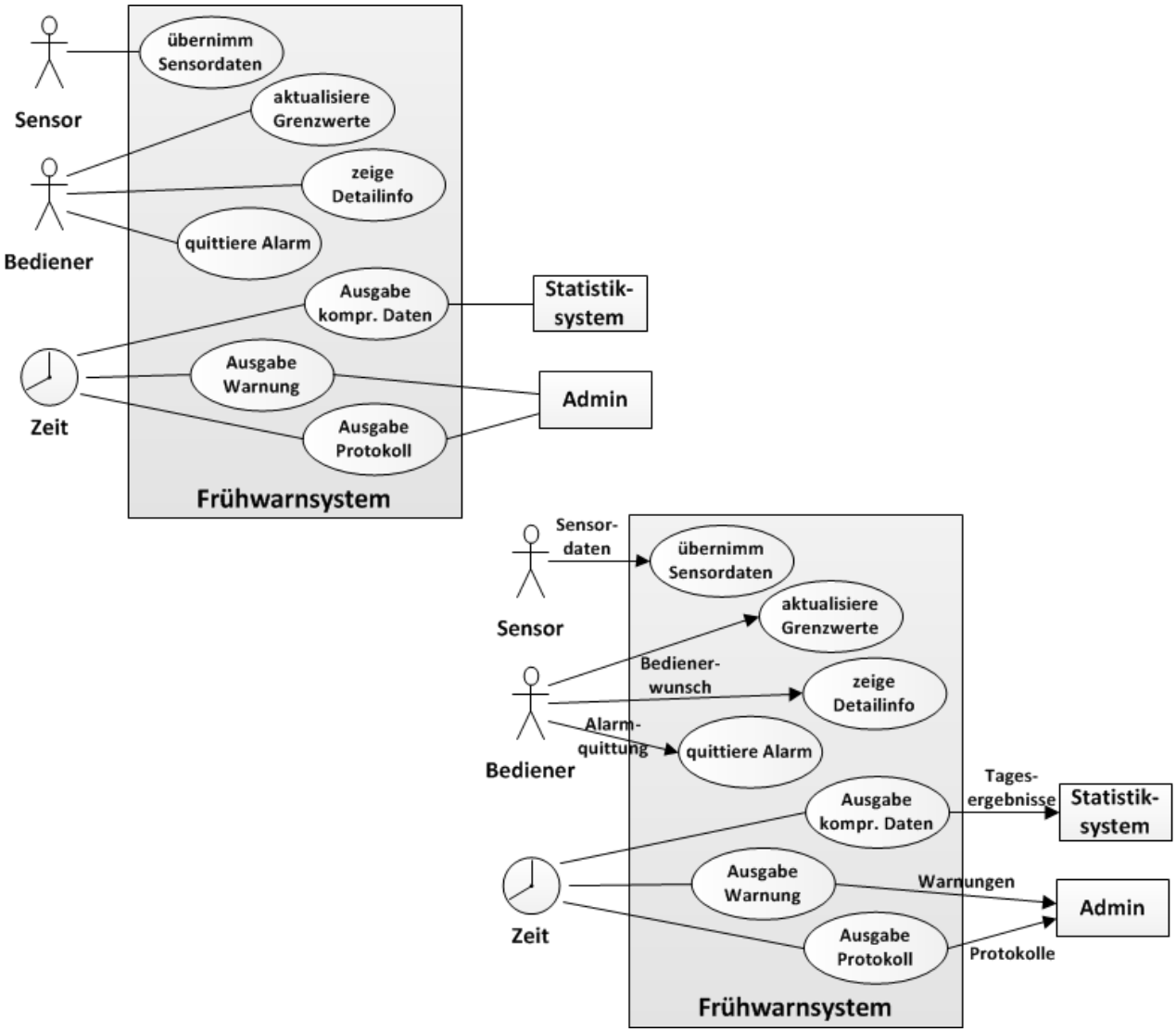

Abbildung 36: (a) Use-Case-Diagramm mit allen Nachbarsystemen; (b) Use-Case-Diagramm mit Ein- und Ausgaben

Empfehlung 1: Nutze die Stärke beider Diagrammarten, um einerseits eine möglichst vollständige Schnittstellenbetrachtung (durch das Kontextdiagramm) zu erhalten und andererseits zu einer Grobgliederung der Funktionalität aus Perspektive der Nutzer (in Form von

Use-Cases) zu kommen, die einen guten Überblick über die geforderte Gesamtfunktionalität bietet und eine separate, ergänzende Spezifikation der einzelnen Use-Cases ermöglicht.

Empfehlung 2: Wenn lediglich Use-Case-Diagramme ohne ein Kontextdiagramm modelliert werden (z. B. weil das eingesetzte Werkzeug keine Kontextdiagramme unterstützt und dieses auch nicht mittels Klassendiagramm ausgedrückt werden soll), dann sollten alle Nachbarsysteme des Systems in den Use-Case-Diagrammen vollständig angegeben sein. Durch die zusätzliche Verwendung grafischer Layoutmöglichkeiten können dann die Auslöser der jeweiligen Use-Cases (d.h. die entsprechenden Akteure) und andere betroffene Nachbarsysteme leicht unterschieden werden (z.B. Akteur links anordnen, andere Nachbarsysteme rechts vom System zeichnen). Selbst ein solch „erweitertes Use-Case-Diagramm" hat allerdings immer noch nicht die Ausdrucksmächtigkeit und Präzision eines Kontextdiagramms, weil in den Use-Case-Diagrammen die Bezeichner der Ein- und Ausgaben fehlen. Diese könnten an die gerichteten Assoziationen zwischen Akteuren und Use-Cases geschrieben werden (siehe Abbildung 36, b), jedoch wird das Diagramm damit sehr überladen und damit schwerer verständlich, was dem eigentlichen Zweck der Use-Case-Modellierung entgegenläuft.

4.2.4 Use-Cases finden

Um die notwendigen Use-Cases des betrachteten Systems zu finden, ist es häufig sinnvoll, sich zunächst auf die Auslöser möglicher Use-Cases zu konzentrieren. Auslöser von Use-Cases sind Ereignisse im Systemkontext, auf die das betrachtete System geeignet reagieren soll, in dem es einen Prozess ausführt, der einem oder mehreren Akteuren im Systemkontext einen fachlichen Mehrwert erbringt. [McPa1984] unterteilen solche Auslöser in zwei Kategorien:

- *Externe Auslöser*: Ein Akteur (z.B. ein Nachbarsystem) möchte einen Prozess in unserem betrachteten System auslösen. Unser System merkt das durch Daten, die vom Nachbarsystem kommen und die Systemgrenze überschreiten. Beispielsweise „Gast wünscht Zimmer in einem Hotelsystem". Sobald der Wunsch eintrifft (d.h. das entsprechende Eregnis im Systemkontext tritt ein), soll das Hotelsystem dem Gast ein geeignetes Zimmer anbieten.
- *Zeitauslöser*: Es ist Zeit, in unserem System einen Prozess auszuführen, z.B. zu bestimmten Uhrzeiten oder an bestimmten Kalendertagen. Zeitereignisse sind dadurch charakterisiert, dass unser System keinen Trigger über die Systemgrenze braucht, um den Prozess zu beginnen, nur das Erreichen des festgelegten Zeitpunkts. Bsp. in dem Hotelsystem: „Es ist 18:00 Uhr und somit Zeit, alle reservierten Zimmer von nichterschienenen Gästen wieder in den freien Verkauf zu geben". Als Zeitereignis wird auch die Überwachung von systeminternen Ressourcen gewertet, wie z.B. „Es ist Zeit, unseren Hotelkatalog nachzudrucken".

4.2.4.1 Durchgängigkeit der Abläufe von Systemgrenze zu Systemgrenze

Im Rahmen der Modellierung von Use-Case-Diagramen sollte darauf geachtet werden, dass die Abläufe – sobald sie extern oder zeitabhängig ausgelöst wurden – bis zu ihrem Ende betrachtet werden und nicht innerhalb des Systems (z.B. anhand bereits bekannter Software-Komponenten oder organisatorischer Zuständigkeiten für einzelne Teile des Systems) unterbrochen werden und dann einen neuen Auslöser benötigen, der den Ablauf wieder anstößt. Die Granularität eines Use-Case wird also durch die **vollständige Reaktion** des betrachteten Systems auf den Auslöser im Systemkontext bestimmt, d.h. durch den vollständi-

gen Ablauf des Use-Cases stellt sich für die primären Akteure der zugehörige fachliche Mehrwert ein.

4.2.4.2 Pragmatische Regeln zur Granularität von Use-Cases: die 80/20-Regel

Im Zusammenhang mit der Use-Case-Modellierung stellt sich oft die Frage nach der geeigneten Granularität für Use-Cases und in welcher Situation mehrere Use-Cases in einem Use-Case zusammengefasst werden sollten. Das Kriterium, dass Abläufe zum gleichen fachlichen Mehrwert führen, ist ein starkes Indiz dafür, dass die entsprechenden Use-Cases zusammengeführt werden sollten. Bei umfangreichen und komplexen Systemen bietet es sich zunächst oft an, die verschiedenen Use-Cases zu untersuchen. Wenn dann zwei Use-Cases 80% identische Abläufe und ähnliche fachliche Mehrwerte für die Akteure haben (z.B. wenn die Abläufe nahezu identisch sind, jedoch mit anderen Daten ausgeführt werden), sollte für diese Abläufe nur ein Use-Case modelliert werden und die Unterschiede der Abläufe in der Use-Case-Spezifikation (siehe Abschnitt 4.2.5) dokumentiert werden. Weisen Abläufe jedoch nur 20% Gemeinsamkeiten auf und geht die Beschreibung mit vielen notwendigen Fallunterscheidungen einher, dann sollten verschiedene Use-Cases modelliert werden. Bei einer „Gleichartigkeit“ von 50% ist eine Entscheidung oft schwierig. Letztlich sollte dann immer der eigentliche Mehrwert, der durch die Abläufe erbracht wird, ausschlaggebend dafür sein, ob mehrere Use-Cases in einem Use-Case zusammengeführt werden oder nicht.

4.2.5 Use-Cases spezifizieren

Die Popularität der Use-Cases ist nicht zuletzt darauf zurückzuführen, dass Ivar Jacobson den Usern „ihre Sprache“ zurückgegeben hat. Er schlägt vor, den gewünschten Ablauf eines Use-Cases in natürlicher Sprache zu beschreiben. Die UML macht keinerlei Vorschläge über den Stil von Use-Case-Beschreibungen. Im Lauf der Zeit wurden viele Vorschläge gemacht, wie die Schwachstellen von rein umgangssprachlichen (d.h. textuelle) Ablaufbeschreibungen von Use-Cases überwunden werden können. Insbesondere in [Cock00] werden die unterschiedlichen Abstraktionsebenen von Use-Case-Beschreibungen und die unterschiedlichen Leserkreise detailliert untersucht.

Die textuelle Spezifikation eines Use-Cases sollte im Wesentlichen die Ein- und Ausgaben (d.h. Daten, siehe auch Kapitel 3) dokumentieren, die in der graphischen Darstellung typischerweise bewusst nicht dargestellt werden. Detailliertere textuelle Use-Case-Spezifikationen sollten darüber hinaus mindestens den geforderten Ablauf und ggf. alternative Abläufe aus Sicht des primären Akteurs beschreiben (d.h. Haupt- und Altenativszenarien, siehe auch Abschnitt 5.2), sowie Vorbedingungen und Nachbedingungen der Use-Case-Auführung angeben, die typischerweise durch Zustände und Zustandsübergänge charakterisiert werden können (siehe Abschnitt 4.4.1). Darüber hinaus sollten mögliche Ausnahmeereignisse und zugehörige Ausnahmeszenarien (siehe auch Abschnitt 5.2) festgehalten werden. Tabelle 2: zeigt ein Beispiel für eine Schablone zur detaillierten textuellen Spezifikation eines Use-Cases.

Abschnitt	Inhalte
ID	Eindeutiger Identifikator des Use-Cases im Entwicklungsprojekt bzw. -programm
Name	Name des Use-Case in Form (identisch mit Namen im Use-Case Diagramm)
Auslöser	Ereignis, das die Ausführung des Use-Cases auslöst
Vorbedingungen	Vorbedingungen, die für die Ausführung des Use-Cases gelten müssen
Nachbedingung	Menge von Nachbedingungen, die nach der erfolgreichen Ausführung des Use-Case gelten
Eingabedaten	Eingabedaten des Use-Case
Ausgabedaten	Ausgabedaten des Use-Case
Ergebnis	Ergebnis des Use-Case, d.h. fachlicher Mehrwert, der den Akteuren durch die Ausführung des Use-Case erbracht wird
Primärer Akteur	Akteur, der den wesentlichen Mehrwert des Use-Case erhält
Weitere Akteure	Akteure, die an der Durchführung des Use-Case beteiligt sind
Hauptszenario	Ablauf (Interaktion), in der der Use-Case zum überwiegenden Teil abläuft (z.B. in 70% aller Fälle). Siehe auch Abschnitt 5.5.1.
Alternativszenarien	Menge alternativer Abläufe die jeweils ebenfalls zur erfolgreiche Ausführung des Use-Case führen (z.B. in 30% aller Fälle). Siehe auch Abschnitt 5.5.2.1.
Ausnahmeszenarien	Menge von Ausnahmeszenarien, d.h. Szenarien, die beim Eintritt einer Ausnahmesituation im Ablauf des Use-Cases ausgeführt werden, um u.a. eine kontrollierte Fehlerbehandlung/Ausnahmenhbehandlung zu ermöglichen, inklusive des zugehörigen Ausnahmeereignisses, d.h. Ausnahmeereignis ⇨ Ausnahmeszenario. Siehe auch Abschnitt 5.5.2.5.

Tabelle 2: Beispiel für eine Schablone zur textuellen Spezifikation von Use-Cases

4.2.6 Use-Cases strukturieren

Die UML stellt drei weitere Ausdrucksmittel zur Verfügung, um die Use-Cases eines Systems zu strukturieren. Abbildung 37 zeigt die Notationen für diese drei Erweiterungen und skizziert kurz deren Bedeutung.

Empfehlung: Es gibt zwar diese Strukturierungsmöglichkeiten in der Syntax der UML. Sie sollten diese Include-, Extend-, Generalisierung-Beziehungen in Use-Case-Modellen jedoch sehr bewusst und eher sparsam verwenden. Dadurch bleiben die Use-Case-Diagramme verständlich und erfüllen damit ihren Zweck. Komplexere Beziehungen im Ablauf eines Use-Cases (d.h. auch zwischen verschiedenen Use-Cases) können oft durch die ergänzende Verwendung von Diagrammen anderen Typs, wie z.B. Aktivitätsdiagrammen (siehe Abschnitt 4.3.3) besser verständlich und präziser ausgerückt werden. Sowohl die Inklusion von Teilprozessen (mittels „Include“) als auch die bedingungsabhängige Erweiterung von Use-Cases durch Teilprozesse (mittels „Extend“) lassen sich in Aktivitätsdiagrammen präziser ausdrücken.

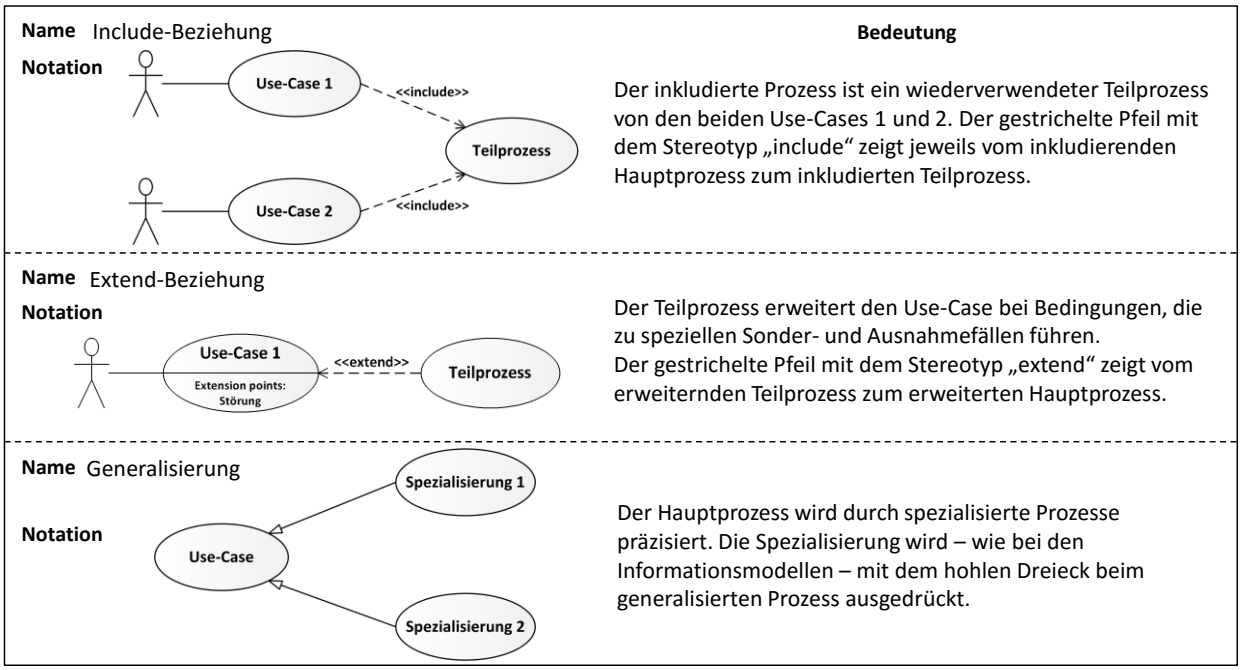

Abbildung 37: Modellierungskonstrukte zur Strukturierung von Use-Cases in Use-Case-Diagrammen

Wenn die obigen Modellierungskonstrukte zur Strukturierung der Use-Cases eines Systems angewendet werden, dann sollten die folgenden Faustregeln berücksicht werden:

- Eine *Include*-Beziehung kann z.B. sinnvoll verwendet werden, um explizit zu dokumentieren, dass mehrere Use-Cases innerhalb eines Use-Case-Diagramms einen identischen Teilprozess besitzen. Dies spart u.a. Mehrfacharbeit in der Spezifikation und ist daher mitunter sehr sinnvoll. Solche Sachverhalte könnten aber auch durch gleichnamige Aktivitäten in den Aktivitätsdiagrammen ausgedrückt werden, die jeweils den Ablauf der Use-Cases dokumentieren, *ohne* dass dadurch das Use-Case-Diagramm umfangreicher und unübersichtlicher wird.
- Eine *Extend*-Beziehung kann z.B. sinnvoll verwendet werden, um zu dokumentieren, dass innerhalb des „normalen“ Ablaufs eines Use-Cases unter einer bestimmten Bedingung zusätzlich ein Teilprozess ausgeführt werden muss. Wichtig ist dabei, dass der Extension Point, d.h. die Bedingung, unter der der Teilprozess zusätzlich ausgeführt wird, möglichst präzise und verständlich formuliert ist. In vielen Fällen ist es allerdings sinnvoll auf die explizite Modellierung einer solchen Extension in Use-Case-Diagrammen zu verzichten, weil die präzise Spezifikation des Extension Points ohnehin nur in der Ablaufbeschreibung modelliert werden kann.
- Mit Hilfe der *Generalisierung* (bzw. *Spezialisierung*) von Use-Cases kann ausgedrückt werden, dass der spezifische Ablauf eines oder mehrerer Use-Cases generalisiert werden kann. In den meisten Fällen werden solche Beziehungen dann modelliert, wenn mehrere Use-Cases in einem Use-Case-Diagramm auftreten, über deren spezifische Abläufe derart abstrahiert werden kann, dass diese auf einer generelleren Ebene gleich sind. Hierzu wird, wie in Abbildung 37 gezeigt, jeweils eine Generalisierungsbeziehung hin zum generalisierten Use-Case modelliert. Erfahrungsgemäß werden in Use-Case-Diagrammen eher selten generalisierte Use-Cases und entsprechende Beziehungen modelliert, da diese Form der Abstraktion eher ein Konzept der Informationsstrukturmodellierung ist, in welcher z.B. über gemeinsame Attribute durch die Bildung von Superklassen abstrahiert wird (vgl. Abschnitt 3). Die Beschreibung abstrakterer (gene-

ralisierter) Prozesse im Vergleich zu ihren konkreten (spezialisierten) Ausprägungen ist im Rahmen der Use-Case-Modellierung meist schwierig. Dieses Modellierungskonstrukt sollte daher nicht leichtfertig, sondern nur wohlüberlegt eingesetzt werden.

4.2.7 Use-Cases paketieren

Für Systeme mit einer großen Zahl von Use-Cases besteht die Möglichkeit, die Darstellung des Use-Case-Modells durch graphische Gruppierung einfacher handhabbar und verständlich zu machen. Wie für viele Elemente erlaubt die UML/SysML es auch für Use-Cases, diese zu paketieren, d.h. eine Menge von Use-Cases in einem Paket zusammenzufassen. Die Kriterien für die Paketierung können frei gewählt werden. Üblicherweise paketiert man logisch zusammengehörige Use-Cases (die z.B. einen ähnlichen fachlichen Mehrwert besitzen), bzw. Use-Cases, die ähnliche Thematiken behandeln (z.B. alle Use-Cases zur Lagerverwaltung in einem ERP-System). Paketierung dient hauptsächlich der Verbesserung der Handhabbarkeit und Lesbarkeit des Use-Case-Modells, wenn eine große Anzahl von Use-Cases für das betrachtete System vorliegt.

4.2.8 Zusammenfassung

Use-Case-Modelle stehen normalerweise nicht am Ende eines systematischen Prozesses der Anforderungsmodellierung, sondern sind für gewöhnlich ein erster sinnvoller Schritt, um die Gesamtkomplexität eines Systems (aus dem Kontextdiagramm heraus) systematisch zu verstehen und zu dokumentieren. Eine zu jedem Use-Case gehörende Use-Case-Spezifikation reicht bei einfachen Prozessen oft als Spezifikation der geforderten Funktionalität aus. Bei komplexen Prozessen sind diese eher der Ausgangspunkt für die Erstellung detaillierterer Anforderungsdiagramme, die das notwendige Verhalten des Systems präzise dokumentieren. Die entsprechenden Diagrammtypen werden in den nächsten Abschnitten vorgestellt.

4.3 Datenfluss- und kontrollflussorientierte Modellierung von Anforderungen

Die Kernelemente der Modelle der dynamischen Sicht sind die geforderten Funktionen, d.h. die Funktionen, die das betrachtete System zur Verfügung stellen soll. In zwei Diagrammtypen wurde diese geforderte Funktionalität bisher erläutert: im **Kontextdiagramm** ist die gesamte Funktionalität als Black-Box dargestellt. In den **Use-Case-Diagrammen** wird die geforderte Funktionalität durch eine Menge von Use-Cases dargestellt.

In diesem Abschnitt werden zwei weitere Diagrammtypen eingeführt, in denen die geforderte Funktionalität des Systems detaillierter und präziser spezifiziert werden kann: Aktivitätsdiagramme der UML und Datenflussdiagramme, wie sie in den Ansätzen zur Strukturierten Analyse (z.B. [DeMa1979]) verwendet werden.

Das Notationselement für Funktionen ist (historisch bedingt) unterschiedlich in den beiden Diagrammen (siehe Abbildung 38), aber der Zweck der beiden Diagramme im Requirements Engineering ist in beiden Fällen gleich: eine Zerlegung der geforderten Funktionalität in kleinere Funktionen und die Diskussion des Zusammenwirkens der Funktionen, um die größere geforderte Funktionalität zu erbringen.

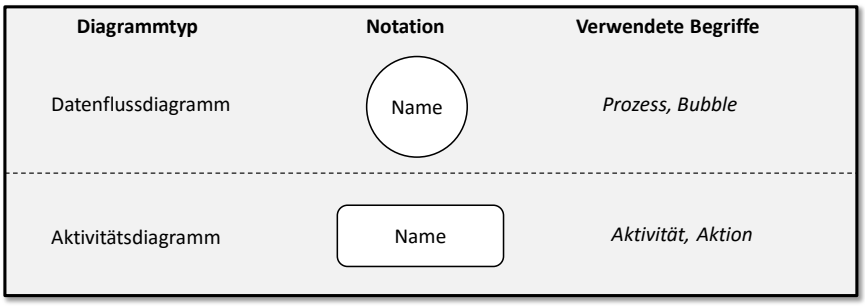

Abbildung 38: Modellierung von Funktionen in Datenfluss- und Aktivitätsdiagrammen

Für das Zusammenwirken von Funktionen gibt es zwei grundlegende Konzepte – Daten- und Steuerfluss, die im nächsten Abschnitt motiviert und erläutert werden. In diesem Handbuch wird je einen Vertreter für „Kontrollflussdenken“ (hier: UML Aktivitätsdiagramme) und ein Vertreter für „Datenflussdenken“ (hier: Datenflussdiagramme) genauer betrachtet. [6]

4.3.1 Zweck

Zu den frühesten Ablaufmodellen in der IT zählen Flussdiagramme (Programmablaufpläne nach DIN 66001). Mit diesen konnte man Funktionen (als Kästchen dargestellt), Alternativen und Verzweigungen (als Rauten) und leider auch beliebige Sprünge (mit Sprungmarken) darstellen. Im Mittelpunkt dieser Sprache stand jedoch, dass man beim Lesen dieser Diagramme „einfach den Pfeilen folgt“, um den spezifizierten Ablauf zu verstehen (vgl. Abbildung 39). Ein Schritt nach dem anderen wird ausgeführt. Diese Pfeile stellten Kontrollfluss dar. Wenn eine Funktion ausgeführt ist, kommt im Ablauf die nächste Funktion entlang der Pfeile dran. Diese Art der Betrachtung steht sowohl bei Aktivitätsdiagrammen in der UML als auch in den Diagrammen der BPMN (vgl. [OMG2011]) im Mittelpunkt. Die Pfeile repräsentierten **Kontrollfluss (Steuerfluss)** und somit einen Ablauf in Form der Ausführungsreihenfolgen der Funktionen. Im Wesentlichen ist immer genau **eine** Funktion aktiv. Wenn die Ausführung einer Funktion abgeschlossen ist, wird die nächste Funktion in der Folge ausgeführt.

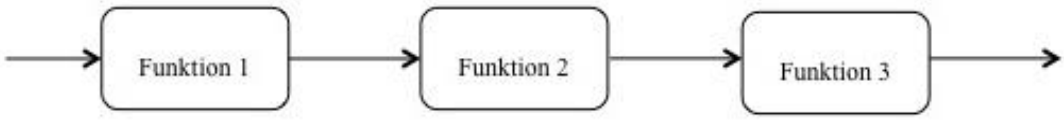

Abbildung 39: Steuerfluss zwischen Funktionen

In den späten 70er-Jahren wechselte mit den Publikationen über “Strukturierte Analyse“ [GaSa1977, DeMa1979, RoSc1977] die vorherrschende Lehrmeinung dahingehend, dass nicht mehr die Steuerflussbetrachtung, sondern Datenflussdiagramme im Mittelpunkt der Analyse-Ansätze standen. Auch Datenflussdiagramme betrachten Funktionen des Systems (dargestellt meist als Kreise, in manchen Notationen auch als Rechtecke mit abgerundeten Ecken oder als Rechtecke), allerdings haben die (beschrifteten) Pfeile zwischen den Funktionen eine andere Bedeutung als in den steuerflussorientierten Ansätzen. Die Pfeile in Datenflussdiagrammen repräsentieren Ein- und Ausgaben der Funktionen, d.h. den **Datenfluss** zwischen den Funktionen und nicht den Steuerfluss (siehe Abbildung 40).

[6] Die UML/SyML unterscheidet auf der Konzeptebene zwischen „Aktion“ und „Aktivität“. Aktionen sind atomare ausführbare Funktionen, d.h. Funktionen, deren Ausführung nicht unterbrechbar ist. Aktivitäten besitzen typischerweise eine gröbere Granularität als Aktionen, können sich aus mehreren Aktionen zusammensetzen und sind (zumindest jeweils nach der Ausführung einer Aktion) unterbrechbar.

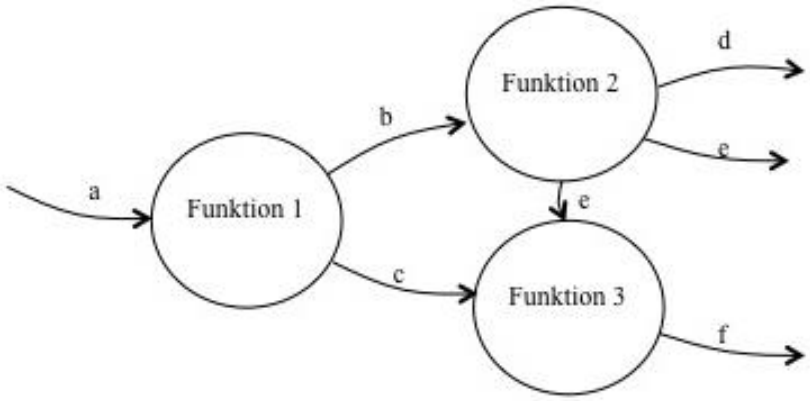

Abbildung 40: Datenfluss zwischen Funktionen

In datenflussorientierten Sichten können im Wesentlichen **alle** Funktionen gleichzeitig aktiv sein. Durch den Datenfluss werden nur kausale Abhängigkeiten festgelegt, d.h. eine Funktion kann frühestens dann arbeiten, wenn ihre Eingaben vorhanden sind. Es wird aber – im Gegensatz zum Steuerfluss – keine explizite Reihenfolge zwischen den Funktionen modelliert. Wenn bei der Verwendung solcher Diagrammen eine explizite Reihenfolgen der Funktionen ausgedrückt werden soll, kann man z.B. einen ergänzenden Zustandsautomaten modellieren, um durch Ereignisse und Zustände eine Reihenfolge der Ausführung von Funktionen auszudrücken. Die Art und Weise des Zusammenwirkens von Datenflussdiagrammen und Zustandsautomaten kann gut anhand der Metapher einer Marionette verdeutlicht werden. Die Funktionen im Datenflussdiagramm entsprechen den Teilen der Puppe (wie Arme, Beine, Kopf), die sich zunächst relativ unabhängig voneinander frei bewegen lassen. Ein Zustandsautomat entspricht dem Holzkreuz mit Fäden zu den beweglichen Puppenteilen. Das Holzkreuz stellt einen (Bewegungs-)Zusammenhang zwischen den beweglichen (veränderlichen) Teilen der Puppe her, wordurch der Marionettenspieler die möglichen Bewegungen der Puppenteile einschränken kann.

Komplexe geforderte Funktionalität kann entweder ablauforientiert (mittels Aktivitätsdiagrammen) oder datenflussorientiert (mittels Datenflussdiagrammen) modelliert werden. Dabei soll nicht die Wahl zwischen den beiden Diagrammarten im Vordergrund stehen, sondern das grundsätzliche Denken in Datenflüssen oder das Denken in Steuerflüssen. Denn: Wie im Folgenden erläutert wird, kann man Datenflussdenken auch in UML-Aktivitätsdiagrammen darstellen und umgekehrt mit Datenflussdiagrammen relativ linear verlaufende Prozesse.

Anmerkung: In einigen Modellierungsansätzen der dynamischen Sicht, wie z.B. in Petri-Netzen, wird vorgeschlagen, Datenfluss und Steuerfluss gemeinsam in den Diagrammen zu modellieren. Dies führt oft zu einer höheren Komplexität der Diagramme und somit zu einer schwereren Verständlichkeit.

4.3.2 Anforderungsmodellierung mit Datenflussdiagrammen (DFDs)

Zur Modellierung von Anforderungen in der datenflussorientierten Sicht werden häufig Datenflussdiagramme eingesetzt. Solche Datenflussdiagramme dokumentieren die Datenabhängigkeiten zwischen notwendigen Funktionen des betrachteten Systems.

4.3.2.1 Modellierungskonstrukte von Datenflussdiagrammen

Abbildung 41 fasst die wesentlichen Modellierungskonstrukte von Datenflussdiagrammen zusammen.

Name	Notation	Bedeutung
Nachbarsystem/ Akteur (auch Terminator, Quelle bzw. Senke)	Name	Stellt Personen, Organisationen oder technische Systeme, Geräte, Sensoren, Aktuatoren aus der Systemumgebung dar, die Quelle und/oder Senke für Informationen in oder aus dem System sind.
Knoten (Prozess, Funktion des Systems)	Name	Stellt eine gewünschte Funktionalität innerhalb des Systems dar.
Datenfluss	Name →	Stellt Daten in Bewegung dar (Eingaben, Ausgaben, Zwischenergebnisse). Dabei kann es sich nicht nur um Informationsfluss, sondern auch um Material- oder Energiefluss handeln.
Datenspeicher	Name	Stellt Daten in Ruhe dar, d.h. Informationen, die über bestimmte Zeiträume festgehalten werden und nicht direkt in andere Funktionen fließen.

Abbildung 41: Modellierungskonstrukte von Datenflussdiagramen

Abbildung 42 zeigt anhand eines Beispiels aus einem Navigationssystem die vier Notationselemente, die in Datenflussdiagrammen zum Einsatz kommen können, und gibt weitere Erläuterungen zur Semantik. Neben den schon besprochenen Funktionen (Processes, Bubbles) und den Datenflüssen sind das Datenspeicher und Terminatoren.

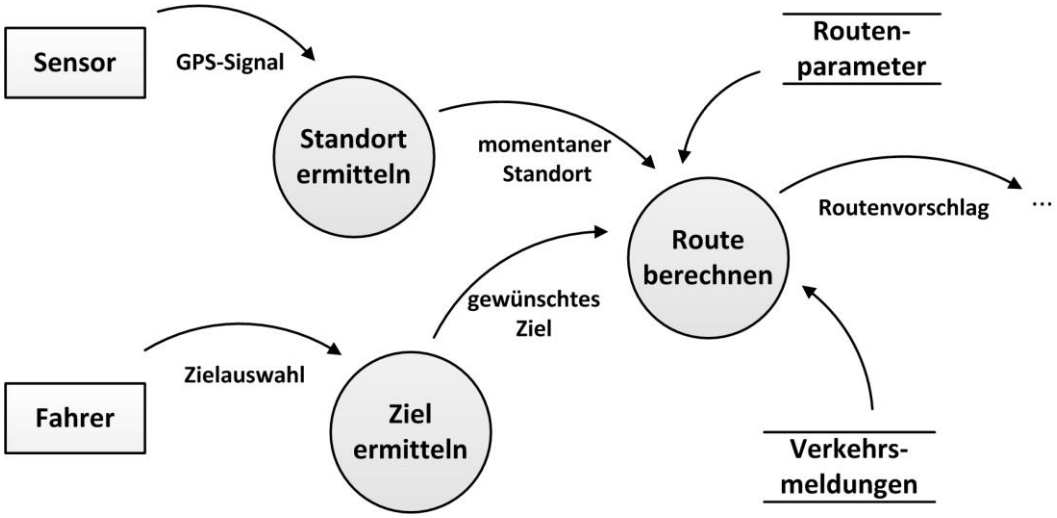

Abbildung 42: Beispiel eines Datenflussdiagramms (Ausschnitt)

Datenflüsse (wie z.B. GPS-Signal, gewünschtes Ziel) sind Daten in Bewegung.

Datenspeicher (wie Routenparameter, Verkehrsmeldungen) sind Daten in Ruhe. Solche Datenspeicher können von Funktionen angelegt und gefüllt sowie von anderen Funktionen (nicht-zerstörend) gelesen werden. Es handelt sich dabei um persistente Daten, wobei der Zeitraum, für den die Informationen (oder Materie oder Energie) festgehalten wird, nicht weiter festgelegt ist. Bei typischen kommerziellen Systemen sind das die Dinge, die technisch in Datenbanken oder Dateien festgehalten werden. Bei technischen Systemen werden die Daten z.B. in Flashspeichern o.ä. festgehalten. Bei organisatorischen Systemen auch in Karteikästen, Aktenordnern oder anderen Speichern.

Das vierte Element (die Rechtecke, im Beispiel „Sensor“ und „Fahrer“) stellen Nachbarsysteme des betrachteten Systems dar. Sie wurden in der Strukturierten Analyse als **Terminatoren** (Terminators) bezeichnet, bzw. als **Quellen** und **Senken** (Sources and Sinks), je nachdem, ob sie Eingaben liefern oder Ausgaben empfangen. Ein Terminator kann sowohl Quelle als auch Senke sein. Diese Terminatoren werden üblicherweise in einem Kontextdiagramm vollständig aufgeführt (vgl. Kapitel 2.2). So gesehen ist das klassische Kontextdiagramm ein spezielles Datenflussdiagramm, in dem **alle** Nachbarsysteme (bzw. Akteure) sowie alle Eingaben und alle Ausgaben modelliert werden, aber die Funktionalität des SuD auf **einen** Knoten beschränkt wird. Wenn die Nachbarsysteme (bzw. Akteure) bereits im Kontextdiagramm aufgeführt sind, werden bei verfeinerten Datenflussdiagrammen häufig keine Terminatoren sondern nur noch die zugehörigen Datenflüsse an der Systemgrenze modelliert (siehe hierzu Abschnitt 4.3.6).

4.3.2.2 Beziehung der Datenflussmodellierung zur Use-Cases, Kontrollfluss- und Informationsstrukturmodellierung

Die datenflussorientierte Modellierung von Anforderungen mit Datenflussdiagrammen besitzt einen wesentlichen Zusammenhang mit dem Kontextdiagramm, der Use-Case-Sicht und der Informationsstruktursicht. Use-Cases stellen grobe Nutzerfunktionalitäten dar, die im Zuge des Requirements Engineerings in feingranularere Systemfunktionen und deren Abhängigkeiten zerlegt werden müssen. Mit Hilfe von Datenflussdiagrammen können die Systemfunktionen eines Use-Cases sowie deren Datenabhängigkeiten untereinander und zu Akteuren (DFD: Teminatoren) im Systemkontext modelliert werden. Die feingranularen Systemfunktionen können dabei durch eine funktionale Analyse der Szenarien eines Use-Cases aufgedeckt werden (siehe hierzu auch Abschnitt 5.5.1.3). Die Struktur der Daten, die innerhalb der Datenflussdiagramme als Datenflüsse (d.h. „Daten in Bewegung“) und Datenspeicher (d.h. „Daten in Ruhe“) modelliert sind, werden in den Diagrammen der Informationsstruktursicht definiert (vgl. Abschnitt 3.1).

4.3.3 Anforderungsmodellierung mit Aktivitätsdiagrammen (ADs)

Zur Modellierung von Anforderungen in der Kontrollfluss-Sicht werden Aktivitätsdiagramme der UML bzw. SysML eingesetzt. Aktivitätsdiagramme dokumentieren die Ablauflogik von Prozessen und werden im Rahmen der Anforderungsmodellierung dazu verwendet, die geforderte Ablauflogik von Prozessen zu spezifizieren, die vom betrachteten System ausgeführt werden müssen, damit dieses im Betrieb seinen Zweck erfüllt.

4.3.3.1 Modellierungskonstrukte von Aktivititätsdiagrammen

Abbildung 43 zeigt die für die Anforderungsmodellierung mit Aktivitätsdiagrammen wesentliche Modellierungskonstrukte zur Modellierung des Kontrollflusses von Prozessen.

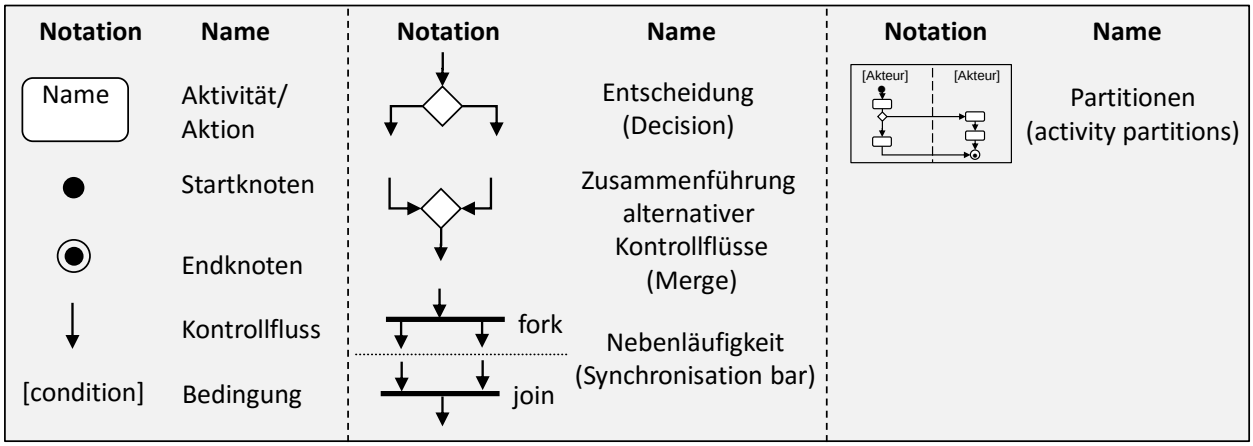

Abbildung 43: Modellierungskonstrukte von Aktivitätsdiagrammen

Neben sequentiellen Abläufen (durch Pfeile dargestellt) lassen sich auch Verzweigungen des Kontrollflusses, aber auch potenziell gleichzeitig ausführbare (genauer: unabhängige/nebenläufige) Funktionen modellieren. Letzteres ist für die Systemanalyse von besonderer Bedeutung, da in realen Systemen viele Dinge gleichzeitig oder unabhängig voneinander passieren können und nicht streng aufeinander folgend. Ein häufig auftretender Modellierungsfehler in Aktivitätsdiagrammen ist die Vorgabe sequentieller Aktivitäten, die in der Realität nicht zwangsweise in dieser Reihenfolge abgearbeitet werden müssen.

Die genaue Syntax und Semantik für die Notationselemente können den einschlägigen Büchern zur UML, wie z.B. [RuJB2004, RuQu2012] entnommen werden. Abbildung 44 illustriert an einem abstrakten Beispiel die typischen Modellierungskonstrukte von Aktivitätsdiagrammen und die wesentlichen syntaktischen Regeln.

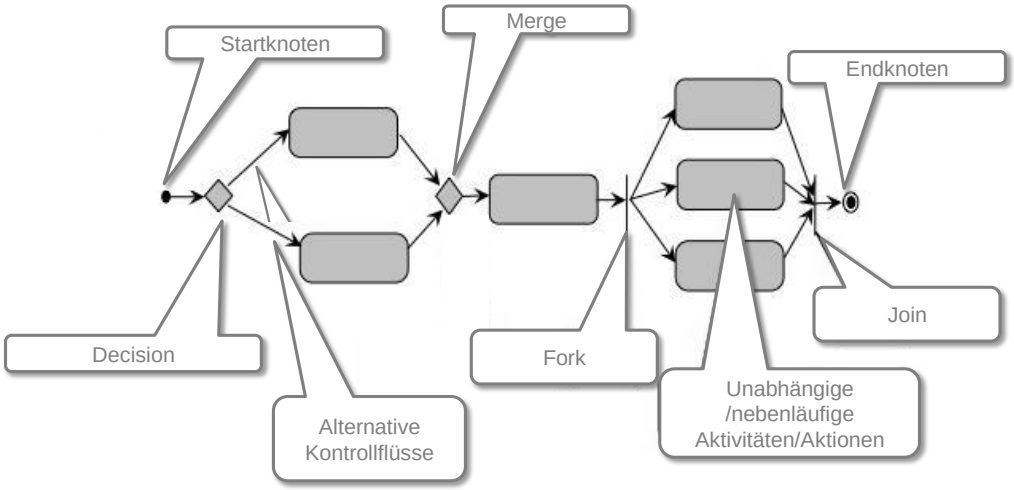

Abbildung 44: Verwendung der Modellierungskonstrukte von Aktivitätsdiagrammen

4.3.3.2 Modellierung von Objekt- und Datenfluss in Aktivitätsdiagrammen und deren Beziehung zur Informationsstrukturmodellierung

Aktivitätsdiagramme bieten auch die Möglichkeit, Daten zu modellieren. Abbildung 45 und Abbildung 46 zeigen Beispiele, die Daten in Bewegung und Daten in Ruhe (Datenspeicher) modellieren. Dies erfolgt durch Einfügen von Objekten (vgl. Abbildung 45) oder Parameter der Aktivitäten (vgl. Abbildung 46). Im Gegensatz zu Datenflussdiagrammen, die fordern, dass alle Ein- und Ausgabe, sowie alle Zugriffe auf Datenspeicher im Diagramm enthalten sind, lassen Aktivitätsdiagramme jedoch offen, wie viele oder wie wenige Daten in den Dia-

grammen erwähnt werden. Bei Datenflussdiagrammen gilt die Grundregel: **alle** eingehenden und **alle** ausgehenden Daten müssen im Diagramm enthalten sein.

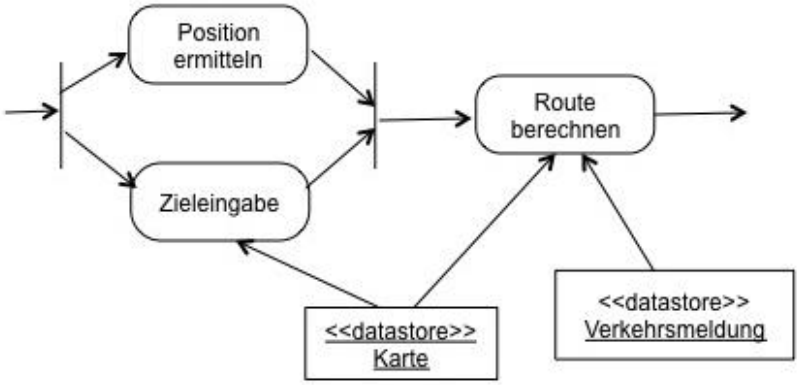

Abbildung 45: Modellierung von Objektfluss in Aktivitätsdiagrammen

Das Beispiel in Abbildung 45 zeigt, dass die Aktivität „Route berechnen“ als Eingabe die Karte und Verkehrsmeldungen benötigt. Es zeigt aber z.B. nicht die wesentliche Ausgabe (die Route oder mehrere Routenvorschläge) und auch nicht eventuell verwendete Routenparameter (wie „schnellste Route“, „kürzeste Route“). Im Gegensatz zu Datenflussdiagrammen, wo extremer Wert auf Vollständigkeit und Konsistenz der Modelle gelegt wird, sollen UML-Diagramme hauptsächlich „nützlich“ für die Kommunikation der Beteiligten sein. Die Vollständigkeit der Spezifikation kann z.B. über hinterlegte Aktivitätsbeschreibungen hergestellt werden.

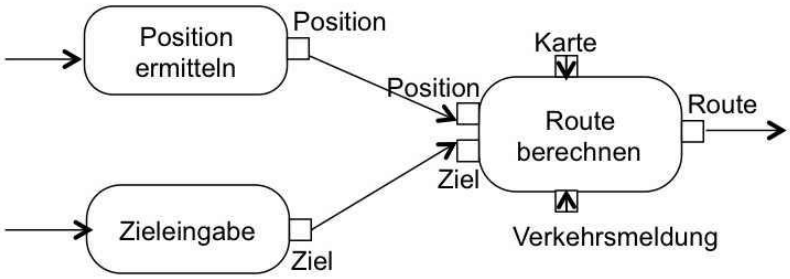

Abbildung 46: Modellierung von Datenflüssen in Aktivitätsdiagrammen durch Pins

Die „Pins“ an den Funktionen stellen die Ein- bzw. Ausgaben der Funktion dar. Somit können Zusammenhänge, wie „Position ermitteln“ erzeugt eine Position als Ausgabe, und „Route berechnen“ benötigt eine Position als Eingabe, grafisch dargestellt werden.

Bei Aktivitätsdiagrammen hat man als Modellierer die Wahl, keinerlei Daten im Diagramm zu modellieren, oder diese gezielt zu modellieren, um bestimmte Aspekte hervorzuheben. Dabei ist allerdings wichtig zu beachten, dass die Ein- und Ausgaben in der Anforderungsspezifikation komplett spezifiziert sein müssen (spätestens bei einer textuellen Spezifikation der einzelnen Funktionen, siehe Abschnitt 4.3.5). Die Struktur der Daten bzw. Klassen und deren Abhängigkeiten zueinander sollten in der Informationsstruktursicht modelliert sein (vgl. Abschnitt 3.1).

4.3.3.3 Beziehung von Aktivitätsdiagrammen zur Use-Case- und Szenariomodellierung

Aktivitätsdiagramme können dazu verwendet werden, die Ablauflogik eines Use-Cases im Detail zu spezifizieren (vgl. Abschnitt 4.2.5). Hierzu werden die durch die Szenarien des Use-Cases induzierten Aktivitäten und deren Reihenfolge im Aktivitätsdiagramm modelliert. Neben der Ablauflogik des Hauptszenarios werden in dem entsprechenden Aktivitätsdiagramm dann die alternativen Abläufe (d.h. Alternativszenarien) modelliert. Dies geschieht typischerweise durch die Verwendung von Entscheidungsknoten, in denen der Kontrollfluss verzweigt und dann abhängig von einer Bedingung entweder die Ablauflogik des Hauptszenarios oder die des Alternativszenarios ausgeführt wird. Abbildung 47 zeigt beispielhaft die Modellierung des Kontrollflusses eines Use-Cases durch ein Aktivitätsdiagramm. Alternativszenarien werden im Kontrollfluss in der Regel durch die Verwendung von Entscheidungsknoten modelliert. In der folgenden Abbildung ist dies z.B. bei den Aktionen „Zieladresse an Tastatur eingeben“ (Teil des Hauptszenarios) und „Zieladresse sprechen“ (Teil des Alternativszenarios) zu erkennen. Ausnahmeszenarien werden oftmals ebenfalls durch die Verwendung von Entscheidungsknoten modelliert. In der obigen Abbildung ist dies beim letzten Entscheidungsknoten zu sehen, der definiert, dass beim Eintreten des Ausnahmeereignisses „Kartenmaterial nicht verfügbar“ in der zugehörigen Ausnahmenbehandlung die Aktionen „Fehlermeldung ausgeben“ und „System abschalten“ durchgeführt werden.

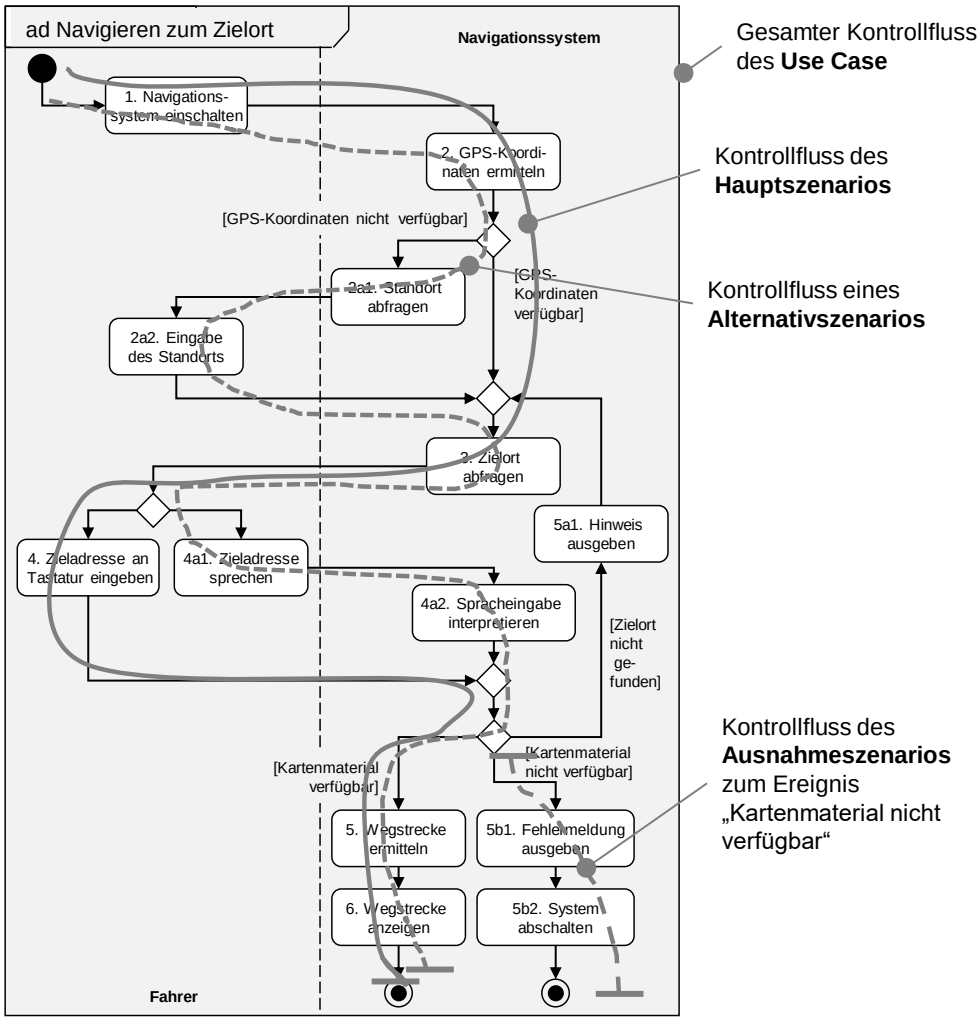

Abbildung 47: Modellierung des Kontrollflusses von Use-Cases durch Aktivitätsdiagramme

Zur Modellierung von Ausnahmenereignissen, die nicht an einer spezifischen Stelle im Kontrollfluss, sondern in einem Teilbereich des Kontrollflusses oder auch beliebig während der Ausführung des Use-Cases auftreten können, werden Signaleingänge und -ausgänge sowie Unterbrechungsbereiche verwendet (vgl. Abschnitt 4.3.7).

4.3.4 Funktionen zerlegen oder bündeln

Beide Diagrammarten (Datenflussdiagramme und Aktivitätsdiagramme) erlauben sowohl das Zerlegen komplexer Funktionen in einfachere Funktionen als auch die Bündelung von einfacheren zu komplexeren Funktionen. In anderen Worten, Datenflussdiagramme und Aktivitätsdiagramme können Hierarchien von Funktionen darstellen (siehe Abbildung 48 und Abbildung 49). Dieser Abstraktionsmechanismus gestattet es, komplexe Sachverhalte zu strukturieren, um sie verständlich und beherrschbar zu halten. Diese Hierarchiebildung ist ein mächtiges Mittel in der dynamischen Sicht der Anforderungsmodellierung, um den Umfang und die Komplexität der betrachteten Systeme im Requirements Engineering beherrschen zu können.

In Abbildung 48 ist die komplexe Funktion „Ziel ermitteln“ eines Navigationssystems verfeinert in 5 (im Beispiel nicht näher benannte) Schritte, wie z.B. „Ziel durch Adresseingabe ermitteln“ oder „Ziel auf Karte festlegen“.

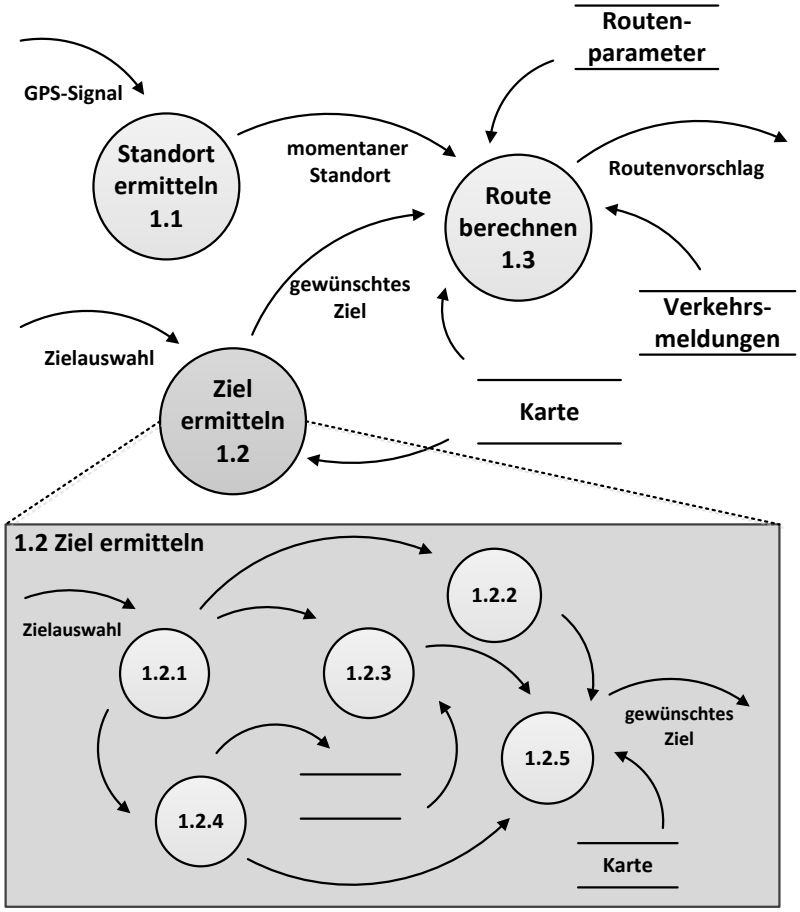

Abbildung 48: Hierarchische Zerlegung bzw. Bündelung von Funktionen in DFDs

In Abbildung 49 ist die komplexe Aktivität B in einen detaillierten Ablauf zerlegt, bestehend aus 5 Schritten. Umgekehrt betrachtet kann man die Aktivitäten B1, B2a1, B2a2, B2b und B3 abstrakt zusammenfassen zu der Gesamtaktivität B.

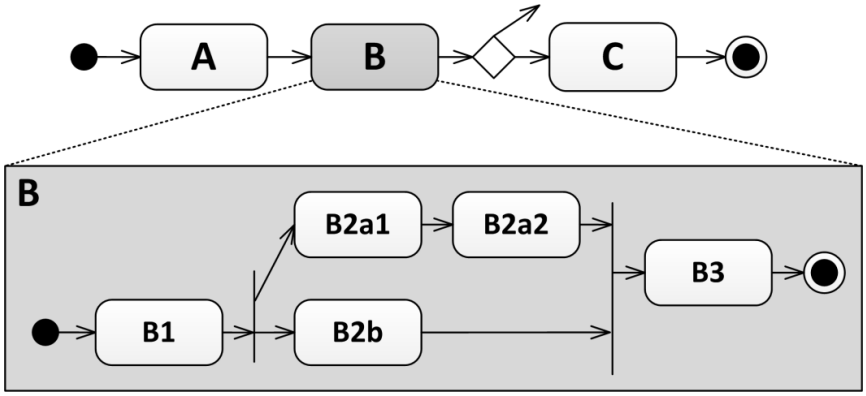

Abbildung 49: Zerlegung einer Funktion in einem Aktivitätsdiagramm

Neben inhaltlichen Kriterien (wie fachlich starker Zusammenhang, was sich oft dadurch äußert, dass man einen guten Namen für die Gesamtheit der Detailaktivitäten findet) kommen sehr pragmatische Kriterien für diese Zerlegung/Bündelung zur Anwendung. So soll das Diagramm meist auf eine Seite eines Dokumentes passen. Deshalb empfehlen die meisten Methoden nicht mehr als 7 +/- 2 Funktionen pro Diagramm zu modellieren.

4.3.5 Textuelle Funktionsspezifikationen

Wie „klein“ sollte man die Funktionen mit Datenflussdiagrammen oder Aktivitätsdiagrammen zerlegen? Oder anders gefragt: Wann sollte man mit der Zerlegung von Funktionen aufhören. Eine einfache Heuristik ist die Länge der für die Funktion notwendigen Beschreibung. Wenn die exakte Spezifikation der Anforderungen an die Funktion noch mehr als eine halbe Seite Beschreibung benötigt, bietet es sich an, eine weitere Verfeinerung des Diagramms in Erwägung ziehen, um zu umfangsreiche natürlichsprachliche Spezifikationen zu vermeiden.

Wenn jedoch das Diagramm bereits alles aussagt, was auszusagen ist, dann hat man wahrscheinlich zu weit zerlegt. Es ist leichter verständlich und lesbar, die letzten ein bis zwei Zerlegungsebenen einzusparen und stattdessen die Verarbeitungssystematik der einzelnen Funktion textuell (z.B. auf einer halben DIN-A4-Seite) zu spezifizieren. Es ist auch möglich, zur Verfeinerung einer Funktion (Aktivität) dieser eine limitierte Menge von 3 bis 7 einfachen natürlichsprachliche Anforderungen zuzuordnen, welche die betrachtete Funktion dann im Detail spezifzieren.

Beispiel: Textuelle Funktionsbeschreibung der Funktion „Ziel ermitteln“ (siehe Abbildung 48)
Funktion: Ziel ermitteln
Eingabe: Zielauswahl (durch den Bediener des Geräts), Karte
Ausgabe: gewünschtes Ziel
Die Funktion soll dem Bediener vier Möglichkeiten geben, ein Ziel auszuwählen:
- *durch Eingabe einer Adresse über die Tastatur*
- *durch Spracheingabe*
- *durch Auswahl aus einer Liste gespeicherter Adressen, oder*
- *durch Auswahl eines Ziels über den Touchscreen der Kartendarstellung*

Für die meisten Verwender dieser Diagramme (insbesondere auch für die Tester, die nach der Fertigstellung des Systems prüfen müssen, ob das System im Betrieb die Anforderungen vollständig und korrekt umsetzt), reicht als abschließende Verfeinerungsebene die oben erwähnte halbe DIN-A4-Seite Spezifikation oft aus, um die Anforderung, die durch diese Funktion ausgedrückt wird, zu verstehen und daraus systematisch Testfälle abzuleiten.

4.3.6 Konsistenzsicherung zwischen Anforderungen verschiedener Abstraktionsebenen des Systems

Im Rahmen einer Anforderungsmodellierung, die Diagramme und textuelle Notationen auf unterschiedliche Abstraktionsebenen (vgl. Abschnitt 4.3.4) umfasst, ist es wichtig, die Anforderungen auf den verschiedenen Abstraktionsebenen konsistent zueinander zu halten. Im Rahmen der Datenfluss-Sicht wurden solche Konsistenzbedingungen in Form sogenannter „Balancing-Regeln“ eingeführt (vgl. [DeMa1979]). Diese **Konsistenzregeln zwischen Diagrammen auf verschiedenen Abstraktionsebenen** sind jedoch auch sinngemäß auf Aktivitätsdiagramme übertragbar:

- Ein- und Ausgaben einer Funktion auf einer Ebene müssen als Ein- und Ausgaben auf der nächstniedrigeren Ebene konsistent vorhanden sein. Dies beginnt beim Kontextdiagramm als abstrakteste Darstellung. Jede Zerlegung des Kontextdiagramms muss alle Schnittstellen, die bereits im Kontext enthalten waren, wieder aufgreifen. Die Ein- und Ausgaben auf der nächsttieferen Ebene müssen jedoch nicht „namensgleich“ sein. Natürlich können auch Daten wie die Funktionen zerlegt werden. So ist z.B. die Ausgabe auf der höheren Ebene „Statistiken“ und auf der nächsttieferen Ebene „Produktstatistiken“, „regionale Statistiken“ und „Verkäuferstatistiken“. Diese Aufteilung wird üblicherweise in einem Glossar (oder Data Dictionary) beschrieben bzw. über die Informationsstruktursicht modelliert. Als Grundregel gilt: Die höhere Ebene darf abstraktere Begriffe enthalten, die bei der Verfeinerung konkretisiert werden.
- Eine besondere Regel gilt für das Balancing von Datenspeicher: Datenspeicher sollten erst auf der Abstraktionsebene eingeführt werden, auf der sie eine Schnittstelle zwischen mindestens zwei Funktionen bilden. In anderen Worten: Ein Datenspeicher, der von einer einzigen Funktion geschrieben *und* gelesen wird, sollte im Inneren der Funktion verborgen werden (also in einer Verfeinerung dieser Funktion) und nicht im dem Diagramm dargestellt werden, wo es nur von einer Funktion gebraucht wird. Ab der Ebene, in der ein Datenspeicher zuerst auftrat, muss der lesende oder schreibende Zugriff auf diesen Datenspeicher auf jeder tieferen Ebene wiederholt werden.

Im Rahmen von Aktivitätsdiagrammen, bei denen Datenflüsse und Datenspeicher nicht notwendigerweise grafisch modelliert werden müssen, sollten die Balancing-Regeln trotzdem beachtet werden. Zur Prüfung muss man zu den Diagrammen auch die ergänzenden Funktionsbeschreibungen heranziehen und inhaltlich überprüfen, ob eine konsistente Verfeinerung der auf verschiedenen Ebenen spezifzierten Anforderungen vorliegt.

4.3.7 Unterbrechungsbereiche und Empfangen/Senden von Nachrichten

Anhand eines Beispiels werden die letzten beiden, für das Requirements Engineering wichtigen Modellierungskonstrukte von Aktivitätsdiagrammen eingeführt: der **Unterbrechungsbereich** und das **Senden von Nachrichten** bzw. das **Empfangen von Nachrichten**:

Beispiel: Ein Nutzer soll die Möglichkeit haben, eine Person auszwählen, für die dann die Kontoumsätze dargestellt werden sollen. Wenn die Umsätze angezeigt werden, hat der Nutzer die Möglichkeit, die Oberfläche zu schließen oder eine neue Person auszuwählen. Auch können neue Umsätze vom System empfangen werden. Die Oberfläche soll sich selbstständig aktualisieren.

In Abbildung 50 ist das gewünschte Verhalten des Systems mit einem Aktivitätsdiagramm modelliert. Der getrichelte Kasten gibt den Unterbrechungsbereich vor. Alle Aktionen, die in

ihm liegen, können bei dem Empfang von Signalen unterbrochen werden (im Beispiel nur die eine Aktion „Kontoumsätze anzeigen“). Wird nun ein Signalempfang innerhalb des Unterbrechungsbereiches modelliert, so wird diese Aktion mit dem Empfang des Signals unterbrochen. Zur besseren Unterscheidung wurden den Signalen die Stereotypen „Benutzeraktion“ und „Systemereignis“ vergeben, um den Auslöser des Signals zu spezifizieren. Nach Empfang eines Signals (und dem Abbruch der aktuell laufenden Aktion) wird bei Bedarf eine Aktion ausgeführt und der Kreislauf kann von vorne beginnen[7] (hier: nach dem Signal „Neuer Umsatz“).

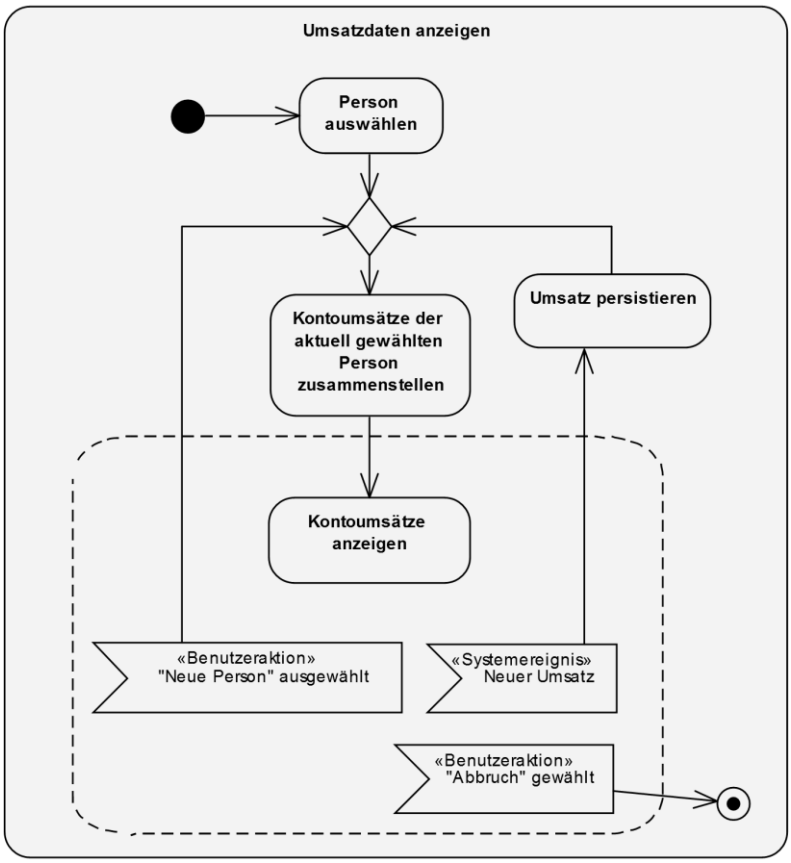

Abbildung 50: Beispiel für die Modellierung von Signalen und Unterbrechungsbereichen

Beendet wird die Aktivität durch die Benutzeraktion „Abbruch gewählt“. Um das gezeigte Diagramm als Anforderungsdiagramm zu vervollständigen, müssten nun die einzelnen Aktionen weiter durch verfeinerte Aktivitätsdiagramme oder durch textuelle Spezifikationen präzisiert werden, um z.B. auszudrücken, wie genau und in welcher Reihenfolge die Umsätze dargestellt werden sollen, oder welche Möglichkeit der Benutzer hat, eine Person auszuwählen.

Neben dem Empfang von Ereignissen können in einem Aktivitätsdiagramm auch Signale erzeugt, d.h. versendet werden. Als Beispiel dient das Aktivitätsdiagramm aus Abbildung 51, durch das jede Sekunde ein Lebenszeichen nach außen gesendet wird (ein sog. Heart-Beat). Erreicht wird dies durch einen „Timer-Event“ (die Sanduhr), der den Ablauf jedesmal für die angegebene Zeit anhält. Auch hier wird wieder ein Unterbrechungsbereich verwendet, um anzuzeigen, wann der Heart-Beat stoppen soll.

[7] Laut UML müssten sogenannte „Unterbrechungskanten“ (Pfeile als Blitz dargestellt) verwendet werden. Wegen Problemen mit der Darstellung im gewählten Tool wurde auf diese Darstellung verzichtet. Die Semantik der Kontrollflusskanten ist an dieser Stelle trotzdem aussagekräftig genug.

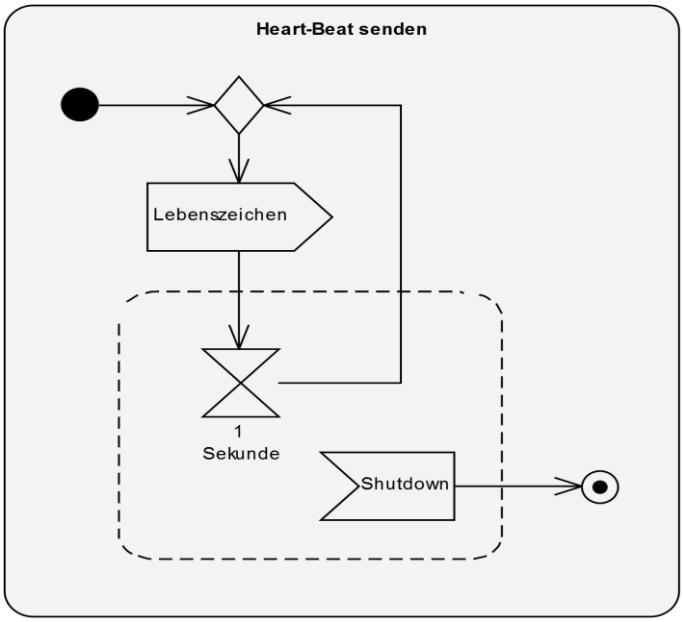

Abbildung 51: Beispiel für ein Heart-Beats

4.3.8 Gegenüberstellung von Datenflussdiagrammen und Aktivitätsdiagrammen in der Anforderungsmodellierung

Die Notation beeinflusst stark unser Denken: In Aktivitätsdiagrammen ist es einfach: „F1 vor F2“ auszudrücken (durch den Pfeil). In Datenflussdiagrammen ist es einfach „F1 erzeugt D und F2 braucht D als Eingabe“ auszudrücken (durch einen beschrifteten Pfeil).

Aktivitätsdiagramme	Datenflussdiagramme
Betonen Kontrollfluss (Ablauflogik) - Sequenzen - Entscheidungen - Fork/Join	Betonen Eingabe-/Ausgabeabhängigkeiten (Datenabhängigkeiten) - Wer erzeugt was? - Wer benötigt was?
Ein- und Ausgaben haben geringere Bedeutung	Kontrollfluss (Ablauflogik) hat geringere Bedeutung
Das Ende einer Aktivität/Aktion triggert die nächste Aktivität/Aktion	Verfügbarkeit der Eingaben ermöglicht die ausführung einer Funktion (Prozess)
Strikter zeitlicher Ablauf (abgesehen von nebenläufigen Kontrollflüssen, d.h. Fork/Join)	Keine implizierte Reihenfolge (außer den kausalen Abhängigkeiten, die durch die Datenabhängigkeiten induziert sind)

Tabelle 3: Unterschiede zwischen der Anforderungsmodellierung mit Datenfluss- vs. Akivitätsdiagrammen

Die Schwerpunktsetzung in den Modellierungssprachen hat sich im Lauf der Jahrzehnte hin und her verlagert. Beginnend mit der Betonung des Kontrollflusses bzw. Steuerflusses (in Flussdiagrammen) zur Betonung des Datenflusses (in DFDs) zurück zur Betonung des Kontrollflusses (in UML Aktivitätsdiagrammen). Beide Konzepte – Kontrollfluss und Datenfluss – sind nützliche Denk- und Ausdruckshilfen zur Spezifikation geforderter Funktionen und deren Abhängigkeiten. Ein Requirements Engineer sollte beide Konzepte kennen und wissen,

wie diese ausgedrückt und modelliert werden können. Durch die derzeitige marktbeherrschende Stellung der UML und dazu passender Tools werden Sie wahrscheinlich als Notation Aktivitätsdiagramme nutzen, sollten aber in der Lage sein, auch in dieser Notation mit Datenfluss und Datenspeichern umzugehen.

4.4 Zustandsorientierte Modellierung von Anforderungen

Mit Hilfe der zustandsbasierten Sicht wird eine weitere dynamische Sicht auf die Anforderungen eines Systems modelliert, die die an der Systemgrenze erkennbaren Zustände und zugehörigen Zustandsübergänge betrachtet. Diese Sicht ist insbesondere für solche Systeme wichtig, deren Verhalten:

- von dem abhängt, was vorher bereits geschehen ist (der Historie) und
- stark von asynchronen Ereignissen beeinflusst wird.

4.4.1 Zweck

Mit dieser Verwendung der Zustandsmodellierung werden im Wesentlichen Vor- und Nachbedingungen spezifiziert, die für das Ausführen einer Funktion (z.B. eines Use-Cases oder einer Aktivität im Aktivitätsdiagramm) gefordert werden. Diese Form der Modellierung lässt sich auf das gesamte System oder auf Teile des Systems anwenden. Ein Teil des Systems kann sich dabei zum Beispiel aus den logischen Komponenten ergeben, nach denen die Use-Cases eingeteilt wurden (siehe Abschnitt 4.2).

Neben der Modellierung der Zustände eines Systems können Zustandsautomaten auch dazu genutzt werden, die Zustände eines Objekts, das im Bereich des Requirements Engineering ein fachliches Objekt aus der Informationssicht (siehe Kapitel 3) darstellt, zu modellieren. Hierbei werden in einem Zustandsautomaten die Auswirkungen auf ein Objekt zusammengefasst, die die verschiedenen Funktionen auf das Objekt bewirken. Gegenüber der rein funktionalen Sichtweise, z.B. in der ablauforientierten Sicht, wird also eine Redundanz eingeführt, die einem der folgenden Zwecke dient:

- Die Konsistenz in der Spezifikation der Funktionen wird überprüft.
- Eine fokussierte Sicht auf ein Objekt erhöht die Verständlichkeit und Nachvollziehbarkeit.

Wichtig bei dem Umgang mit Zustandsautomaten ist die bewusste Festlegung des Betrachtungsgegenstandes, für den die Zustände modelliert werden. Dabei kann es sich um einen der folgenden Betrachtungsgegenstände handeln:

- das betrachtete System
- Teilsysteme des Systems
- die Objekte einer Klasse aus der Informationssicht.

4.4.2 Zustandsbegriff

Der Zustandsbegriff, wie er im Allgemeinen im Requirements Engineering verwendet wird, leitet sich aus der Automatentheorie ab: Ein Zustand ist eine Zusammenfassung von bestimmten Bedingungen, die während eines Zeitintervalls für einen Betrachtungsgegenstand gelten.

Doch wie leiten sich die „Bedingungen“ für einen Betrachtungsgegenstand her?

Wird ein Objekt (eine Instanz einer Klasse) betrachtet, so leiten sich die möglichen Zustände aus allen möglichen Kombinationen der Belegungen seiner Attribute her[8]. So sind in dem in Abbildung 52 gezeigten Beispiel für ein Auto sechs Belegungen für die Farbe, und zwei Belegungen für das Attribut „fahrbereit“ möglich. Damit ergeben sich insgesamt 12 potenzielle Zustände für ein Auto.

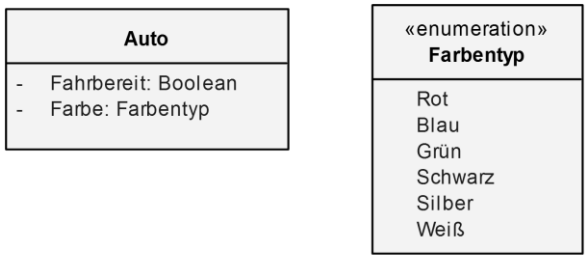

Abbildung 52: Definition eines Autos (a)

Erweitert man das obige Beispiel um ein weiteres Attribut, das die Laufleistung angibt, so stößt man auf ein Problem, wenn dieses Attribut unendlich viele Werte annehmen kann (siehe Abbildung 53). Die Anzahl der potenziellen Zustände wird damit auch unendlich, und diese lassen sich nicht mehr graphisch und in Form eines endlichen Zustandsautomaten darstellen.

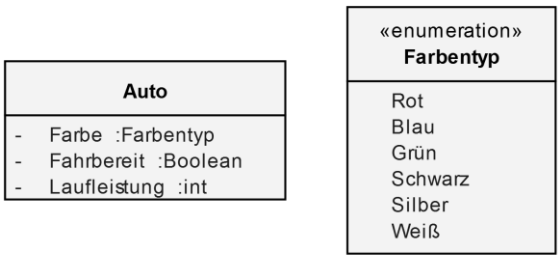

Abbildung 53: Definition eines Autos (b)

Methoden zur Reduktion der Anzahl der Zustände auf ein handhabbares Maß werden in Abschnitt 4.4.4 vorgestellt.

In der Anforderungsmodellierung verwendet man im Allgemeinen nicht die aus der Theorie bekannten Endlichen Automaten (bzw. Moore- oder Mealy-Automaten), sondern die von Harel 1987 eingeführten Statecharts [Hare1987] bzw. die Erweiterung der Harel-Statecharts in der OMG UML [OMG2010b, OMG2010c] und der OMG SysML [OMG2010a]. Die Harel-Statecharts unterscheiden sich von den ursprünglichen endlichen Automaten im Wesentlichen in den drei folgenden Punkten, die die Modellierung in der zustandsbasierten Sicht im Requirements Engineering erheblich vereinfachen:

- Ausgiebigere Möglichkeiten zur Anbindung von Funktionen an Zustände bzw. an Zustandsübergänge (Transitionen)
- Einführung von Bedingungen (Guards), die z.B. bei Transitionen gelten müssen

[8] Diese Betrachtung zur Herleitung der Zustände kann auf jede Art von Betrachtungsgegenständen übertragen werden, da auch z.B. ganze Systeme Eigenschaften besitzen, die als Attribute angesehen werden können.

- Einführung der Möglichkeit von hierarchischen Zustandsautomaten und orthogonalen Regionen

Gerade der zweite Punkt hat sehr große Auswirkungen auf die Modellierung in der zustandsbasierten Sicht, da nicht mehr die gesamte zu betrachtende Historie in Form von Zuständen modelliert werden muss und damit die Anzahl der betrachteten Zustände und die Komplexität der erstellten Diagramme sinkt.

Eine Eigenschaft ist allen Zustandsautomaten gemein: Der Betrachtungsgegenstand des Zustandsautomaten befindet sich zu einem Zeitpunkt immer in einem definierten Zustand. Dies impliziert, dass die Transitionen zwischen zwei Zuständen keine zeitliche Ausdehnung haben. In einer Realisierung, z.B. in Software, benötigen diese Übergänge jedoch Zeit. Wenn man von dieser sehr kurzen Zeitspanne abstrahiert, lässt sich die eingangs dieses Absatzes ausgedrückte Forderung etwas weicher formulieren: Ein Objekt darf auf Ereignisse von außen nur dann reagieren, wenn es sich in einem definierten Zustand befindet. In Bezug auf die Realisierung bedeutet dies, dass die eingehenden Ereignisse für die kurze Dauer der Transition gepuffert werden müssen. Dadurch wird dann die geforderte Semantik eines Zustandsautomaten sichergestellt.

4.4.3 Ein einfaches Beispiel

Das in Abbildung 54 gezeigte Diagramm enthält einen vereinfachten Zustandsautomaten für ein Scheibenwischersystem in Fahrzeugen. Im Beispiel sind einige der wichtigsten Modellierungskonstrukte zur Modellierung in der zustandsbasierten Sicht dargestellt. Sie werden in den folgenden Abschnitten zusammen mit den Notationselementen der UML genauer vorgestellt.

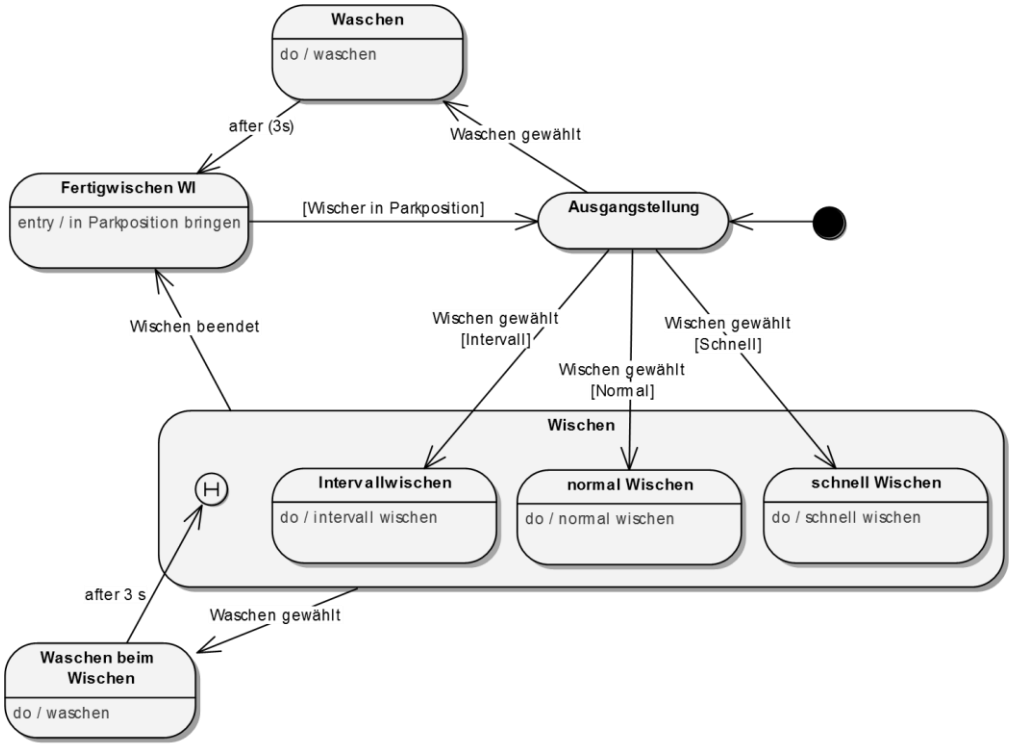

Abbildung 54: Zustandsdiagramm für ein Scheibenwischersystem

4.4.4 Modellierungskonstrukte von Zustandsmaschinendiagrammen

In diesem Abschnitt stellen wir die in der Anforderungsmodellierung gebräuchlichsten Modellierungskonstrukte zur Modellierung in der zustandsbasierten Sicht kurz dar. Dabei verwenden wir die Notationsmittel der UML. Für weitere Notationsmittel und Erklärungen wird auf [OMG2010b, OMG2010c] und [RuQu2012] verwiesen.

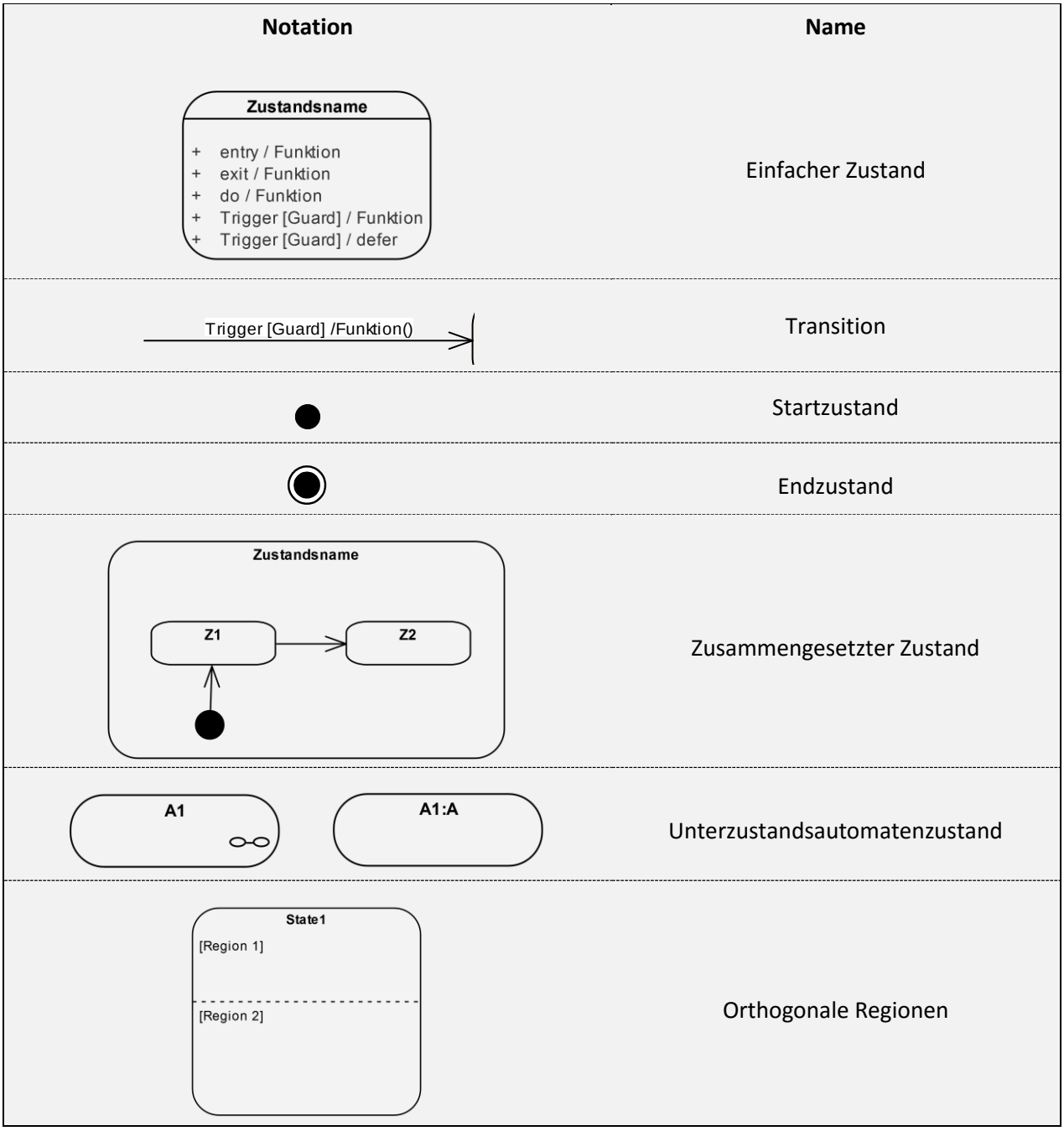

Abbildung 55: Modellierungskonstrukte von Zustandsautomaten (Ausschnitt)

4.4.4.1 Einfacher Zustand

4.4.4.1.1 Syntax und Semantik

In der UML wird ein solcher Zustand mit dem Notationselement wie in der folgenden Abbildung dargestellt.

Abbildung 56: Notation eines Zustands

Ein Zustand sollte immer einen Namen besitzen. Zusätzlich können Sie angeben, welche Funktionen wann in diesem Zustand aufgerufen werden. In der UML sind die folgenden Arten von Funktionsaufrufen in einem Zustand definiert. Die *kursiv* geschriebenen Bezeichner sind in der UML definierte Schlüsselwörter mit einer definierten Semantik. Der Bezeichner „Funktion“ steht für diejenige Funktion, die in dem jeweiligen Fall dann ausgeführt wird.

- Eintrittsverhalten: *entry* / Funktion: Beim Betreten eines Zustandes wird die Funktion ausgeführt. Sie kann nicht unterbrochen werden.
- Austrittsverhalten: *exit* / Funktion: Beim Verlassen eines Zustandes wird die Funktion ausgeführt. Sie kann nicht unterbrochen werden.
- Zustandsfunktion: *do* / Funktion: Während sich der Betrachtungsgegenstand in dem Zustand befindet, wird die Funktion ausgeführt. Sie kann durch Eintreffen eines Triggers, der zu einem Zustandswechsel führt, unterbrochen werden.
- Ausgelöste Funktion: Trigger [Guard] / Funktion: Beim Eintreffen des Triggers und falls die Guard zutrifft, wird die Funktion ausgeführt, ohne dass der Zustand verlassen wird[9].
- Verzögerung: Trigger [Guard] / *defer:* Wenn ein Ereignis in der Defered-Event-Liste des aktuellen Zutandes steht wird dieses Ereignis solange aufgeschoben bis ein Zustand erreicht wird, der dieses Ereignis nicht in der entsprechenden Liste hat. In diesem Zustand dann kann das Ereignis ein Trigger (z.B.) für einen spezifischen Zustandsübergang sein (siehe auch Abschnitt 4.4.4.2).

Für die Zustände gelten die folgenden Regeln:

- Ein Zustand wird betreten, wenn eine Transition, die ihn als Endpunkt besitzt, durchlaufen wird (siehe Abschnitt 4.4.4.2).
- Ein Zustand wird verlassen, wenn eine Transition, die von ihm weg führt, durchlaufen wird.
- Ein Zustand wird aktiv, sobald er betreten wird. Durch Verlassen des Zustands wird der Zustand inaktiv.
- Sofort nach Betreten eines Zustands wird dessen Eintrittsverhalten (Entry-Verhalten) ausgeführt (hier: Funktion 1). Entsprechend wird beim Verlassen des Zustands als abschließendes Funktion (hier: Funktion 2) das Austrittsverhalten (Exit-Verhalten) ausgeführt.
- Das Zustandsverhalten eines Zustands (Do-Verhalten) ist die Funktion (hier: Funktion 3), die nach Beendigung des Eintrittsverhaltens aufgerufen wird.

[9] Damit wird weder die Do-Funktion unterbrochen, noch werden die Entry- und Exit-Funktionen ausgeführt.

- Ein Zustand kann erst über eine Transition verlassen werden, nachdem das Entry-Verhalten (hier: Funktion 1) vollständig ausgeführt wurde.
- Das Auslösen von Funktion 4 durch einen Trigger und einer etwaigen Bedingung führt nicht zum Verlassen des Zustands, auch dann nicht, wenn, wie gezeigt, eine Funktion (hier: Funktion 5) in einem Zustand verzögert wird (defered).

4.4.4.1.2 Finden von Zuständen

Unter dem rein theoretischen Gesichtspunkt aus Abschnitt 4.4.2 folgen im Allgemeinen für ein Objekt sehr viele, manchmal sogar unendlich viele Zustände. Um diese Anzahl von Zuständen auf ein sinnvolles Maß zu verringern, werden zwei Vorgehensweisen empfohlen:

- Für die Zustandsbetrachtung irrelevante Attribute weglassen
- Äquivalenzklassen der möglichen Attributwerte bilden

Betrachtet man das Beispiel aus der Einleitung, so kann man sich überlegen, ob für die betrachtete Aufgabenstellung bei dem Objekt *Auto* das Attribut *Farbe* Relevanz besitzt. Falls nicht, so muss es für die Zustandsbetrachtung auch nicht hinzugezogen werden.

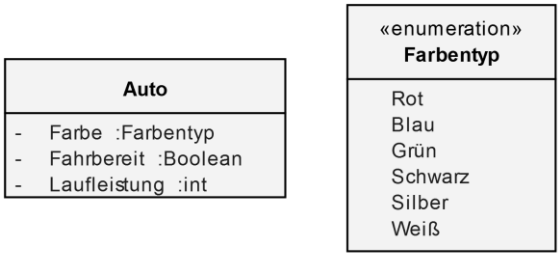

Abbildung 57: Definition eines Autos (c)

Für die Einführung von Äquivalenzklassen entscheidet man, ob die möglichen Werte der Attribute in bestimmte Bereiche eingeteilt werden können. Für die Bereiche muss gelten, dass sich der Betrachtungsgegenstand gleich verhält bzw. verhalten soll, unabhängig davon welcher Wert aus einem Bereich für die Belegung eines Attributs gewählt wird. So kann es sinnvoll erscheinen, die Laufleistung in die drei Bereiche „wenig“, „mittel“, „viel“einzuteilen. Damit reduziert sich die Anzahl der theoretischen Zustände auf eine endliche Anzahl.

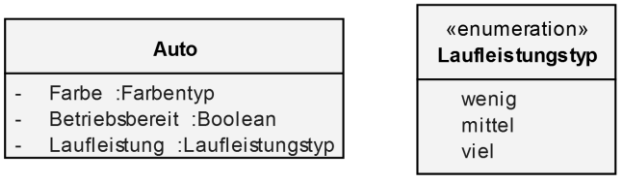

Abbildung 58: Definition eines Autos (d)

Die Zahl der so entstehenden Zustände kann durch fachlich sinnvolles Zusammenfassen von Zuständen weiter verringert werden.

Bei der Betrachtung von Systemen werden Zustände nach der folgenden Regel identifiziert: Systemzustände unterscheiden sich untereinander dadurch, dass das betrachtete System abhängig davon, in welchem Systemzustand es sich gerade befindet, ein unterschiedliches Verhalten nach außen aufweist. Diese Unterschiede spiegeln sich meistens in der Tatsache wider, dass einem Akteur je nach Zustand unterschiedliche Funktionen zur Verfügung gestellt werden.

4.4.4.2 Transitionen

4.4.4.2.1 Syntax und Semantik

In der UML wird eine Transition durch einen Pfeil mit einer entsprechenden Benennung dargestellt. Er verbindet einen Ausgangszustand mit einem Zielzustand.

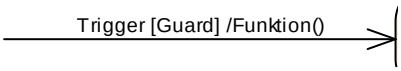

Abbildung 59: Notation einer Transition

Dabei besteht die Benennung der Kante aus den folgenden optionalen Elementen:

- *Trigger*: der Auslöser für die Transition. Die einzelnen Trigger werden durch Kommata voneinander getrennt.
- *Guard*: eine Bedingung, die wahr sein muss, damit die Transition bei Erhalt des Triggers durchlaufen wird. Die Guard-Bedingung wird in eckigen Klammern notiert.
- *Funktion*: Die Funktion, die beim Durchlaufen der Transition ausgeführt wird.

Zu beachten ist hier, dass laut Definition das Durchlaufen der Transition keine Zeit in Anspruch nehmen darf. Es sollten hier also nur „kurze“ Funktionen referenziert werden[10] (wie z.B. das Starten oder Stoppen eines Motors).

Normalerweise wird der Ausgangszustand mit dem Durchlaufen einer Transition verlassen und ein anderer Zustand als Zielzustand erreicht. Jedoch kann es sich bei Ausgangs- und Zielzustand um denselben Zustand handeln. Diese spezielle Art der Transition wird als Selbsttransition bezeichnet (self transition).

Die Transitionen werden durch Trigger ausgelöst und durchlaufen, wenn die entsprechende Guard den Wert „wahr“ besitzt. Dies gilt natürlich nur dann, wenn auch eine Guard an der Transition spezifiziert ist.

Die UML unterschiedet zahlreiche Arten von Triggern. Im Rahmen der Anforderungsmodellierung sind hauptsächlich die beiden Folgenden anzutreffen:

- *SignalTrigger*: Ein SignalTrigger ist ein in den aktiven Zustand eingehendes Signal, das die Ausführung einer Transition auslöst. Deswegen wird der Begriff „trigger“ und Signal“ sehr häufig synonym verwendet.
- *TimeTrigger*: Mit einem TimeTrigger können Sie eine Transition zu einer bestimmten Zeit oder nach einer bestimmten Zeitdauer auslösen. In der OMG UML/SysML dient dazu das Schlüsselwort AFTER, das anstelle des Namens an der Transition notiert wird.

Neben dem Auslösen durch einen Trigger kann eine Transition auch ohne Trigger durchlaufen werden. Dies ist dann der Fall, wenn Sie nur eine Guard und keinen Trigger an der Transition notiert haben, und sich der Wert der Guard von „falsch“ nach „wahr“ ändert.

Eine Guard kann sowohl auf bestimmte Werte wie zum Beispiel „x = 5“ oder Wertebereiche „x >= 10“, als auch auf Aussagen wie „x befindet sich auf der Arbeitsoberfläche“ auf Gültigkeit prüfen („x“ kann hierbei einen von einer Operation oder einem Signal übergebenen Parame-

[10] In einem Softwaresystem wird jede Aktion, auch das Setzen eines neuen Zustands, Zeit in Anspruch nehmen. Hier muss definiert werden, wie diese Nullzeit beim Ändern des Zustands eines Gegenstands realisiert wird, damit nach außen hin die Bedingung der Nullzeit gilt. Dies kann z.B. dadurch gesichert werden, dass der Gegenstand nur auf Ereignisse reagiert, wenn der Zustandsübergang abgeschlossen wurde.

ter oder eine Systemvariable darstellen). Entscheidend ist, dass die Guard eine boole'sche Bedingung repräsentiert. Der Wahrheitswert dieser Bedingung kann zu jedem Zeitpunkt ausgewertet werden, d.h. die Bedingung besitzt zu jedem Zeitpunkt entweder den Wert „wahr" oder „falsch".

Das Empfangen eines Signals und das daraus folgende Auslösen einer Transition wird nur dann ausgeführt, wenn sich der Betrachtungsgegenstand in einem Zustand befindet, von dem eine Transition wegführt, die das Signal als Trigger beinhaltet. Ist für den aktuellen Zustand keine solche Transition definiert, so wird das Signal verworfen. Es sei denn, im aktuellen Zustand ist dieses Signal als „zu verzögern" (defer) definiert. Dann wird dieses Signal zurückgestellt und nach dem nächsten eingetroffenen Signal erneut verwertet.

Transitionen schaffen einen Übergang von einem Ausgangs- zu einem Zielzustand. Dabei sollten sich zwei Transitionen, die denselben Ausgangszustand besitzen, durch verschiedene Trigger oder bei gleichem Trigger durch verschiedene Guards unterscheiden. Dies ist zwar keine Voraussetzung, macht aber die Abarbeitung des resultierenden Zustandsautomaten deterministisch[11].

4.4.4.2.2 Finden von Transitionen

Für das Finden der Transitionen gibt es zwei unterschiedliche Vorgehensweisen:

- Identifikation von Transitionen über ausgehende Zustände
- Identifikation von Transitionen über eingehende Signale.

Die erste Möglichkeit ist sehr intuitiv, da man sich bei der Identifikation der Zustände schon Gedanken gemacht hat, warum zwei Zustände unterschieden werden sollen, und wann von einem Zustand in den anderen gewechselt werden soll. Ähnlich gehen Sie z.B. auch dann vor, wenn Sie die Use-Cases, die Sie den Zuständen als Funktion zugeordnet haben, untersuchen. Ist die dort formulierte Nachbedingung als Zustand definiert? Falls ja, dann muss eine Transition in diesen Nachfolgezustand hinein existieren, da das System nach Beendigung des Use-Cases genau diesen Zustand einnehmen soll (siehe auch Abschnitt 4.2).

Die zweite Möglichkeit ist eher methodisch. Hierbei wird überlegt, ob und wenn ja wie der Use-Case auf das Eintreffen eines Signals von außen reagieren soll, wenn sich das betrachtete System in einem bestimmten Zustand befindet. Dies wird für alle potenziell eintreffenden Signale und für alle Zustände wiederholt. Diese Vorgehensweise kommt eher bei der Betrachtung eines technischen Systems in Frage, bei dem vielleicht schon die Schnittstellen zu den externen Nachbarsystemen vorgegeben sind.

Die zweite Art des Findens von Transitionen ist auch eng mit der Modellierung in der Szenariosicht (siehe Kapitel 5) verknüpft. Eine Nachricht, die dort von einem Betrachtungsgegenstand empfangen wird, führt im Allgemeinen während der Bearbeitung der Nachricht zu einem oder mehreren Zustandsübergängen. Somit dient eine Modellierung in der Szenariosicht auch zum Finden und Verifizieren der Zustandsübergänge in einem Zustandsautomaten.

[11] Deterministisch bedeutet, dass der betrachtete Gegenstand sich z.B. bei gleichen Ausgangssituationen immer gleich auf den Empfang eines Ereignisses verhält.

4.4.4.3 Startzustand

4.4.4.3.1 Syntax und Semantik

Abbildung 60: Notation eines Startzustandes

Wann immer ein Zustandsautomat startet, wird als erste Transition die Transition durchlaufen, die von einem Startzustand in einen Zustand führt. Da ein System sich immer in einem definierten Zustand befinden muss, wird der Startzustand auch als „Pseudozustand“ bezeichnet. Das System befindet sich zu keinem Zeitpunkt in einem solchen Zustand. Damit darf auch keine Guard und kein Trigger an der Ausgangskante eines Startzustands notiert werden.

Neben einem Startzustand in einem Zustandsautomaten, können sich auch Startzustände in den zusammengesetzten Zuständen befinden, mehr dazu in Abschnitt 4.4.4.5.

4.4.4.3.2 Finden von Startzuständen

Jeder Zustandsautomat sollte genau einen Startzustand besitzen. Ihn zu finden ist folglich nicht schwer. Sie zeichnen damit den ersten Zustand, in dem sich das System nach dem Start befinden soll.

4.4.4.4 Endzustand

4.4.4.4.1 Syntax und Semantik

Abbildung 61: Notation eines Endzustands

Wenn der Endzustand erreicht wird, ist die Abarbeitung des umfassenden Zustandsautomaten beendet. In einem Endzustand wird kein weiteres Verhalten ausgeführt. Deshalb darf es bei Endzuständen keine ausgehenden Transitionen geben. Der Endzustand kann fachlich mit dem Ende des Lebenszyklus des modellierten Betrachtungsgegenstandes gleichgesetzt werden.

4.4.4.4.2 Finden von Endzuständen

An dieser Stelle muss die Fachlichkeit des Betrachtungsgegenstands genau analysiert werden. Welcher Lebenszyklus ist für Ihre Anforderungen relevant? Wird z.B. eine Software ausschließlich zu ihrer Laufzeit betrachtet, so wird mit dem Endzustand die Beendigung der Software gleichgesetzt. Falls aber ein eingebettetes System über den gesamten Zeitraum, in dem es in seiner Umgebung „eingebaut“ ist, betrachtet wird, wird kein Endzustand benötigt, weil das System regulär ggf. niemals terminiert (siehe auch Beispiel in Abschnitt 4.4.3).

Darüber hinaus existieren Endzustände auch in den zusammengesetzten Zuständen, die in dem nächsten Abschnitt vorgestellt werden.

4.4.4.5 Zusammengesetzter Zustand

Zusammengesetzte Zustände setzen sich aus einem oder mehreren Zuständen zusammen.

4.4.4.5.1 Syntax und Semantik

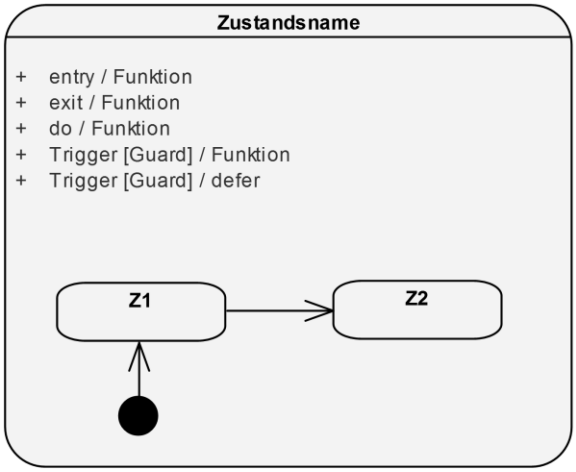

Abbildung 62: Notation eines zusammengesetzten Zustandes

Die enthaltenen Zustände werden als Unterzustände bezeichnet. Als Unterzustände eines zusammengesetzten Zustands sind alle Arten von Zuständen möglich. Dies bedeutet, dass Sie neben den einfachen Zuständen und Pseudozuständen wiederum zusammengesetzte Zustände benutzen können, wodurch Sie eine Hierarchie von Zuständen definieren. Die Blätter in dem entstehenden Zustandsbaum sind dabei die einfachen Zustände, die inneren Knoten sind zusammengesetzte Zustände.

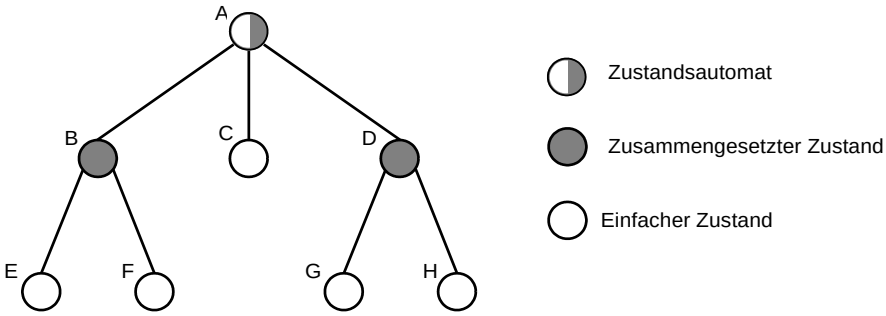

Abbildung 63: Hierarchische Zustände

Die Wurzel des Zustandsbaumes bildet eine Ausnahme, da sie in einem vollständig definierten Modell immer einen Zustandsautomaten darstellt. Dieser beschreibt dann, von außen betrachtet, die Verhaltensbeschreibung des Betrachtungsgegenstands.

Wie in Abschnitt 4.4.2 erwähnt, muss zu jeder Zeit gelten, dass immer genau ein Zustand in einem Zustandsautomaten aktiv sein muss. Wenn dieser Zustand ein zusammengesetzter Zustand ist, ist einer seiner Unterzustände aktiv. Da dieser Unterzustand wiederum ein zusammengesetzter Zustand sein kann, pflanzt sich die Definition der aktiven Zustände in der Hierarchie nach unten fort, bis ein einfacher Zustand als aktiv bezeichnet werden kann.

4.4.4.5.2 Betreten zusammengesetzter Zustände

Für das Betreten eines solchen Zustandes können die Möglichkeiten aus der folgenden Abbildung unterschieden werden.

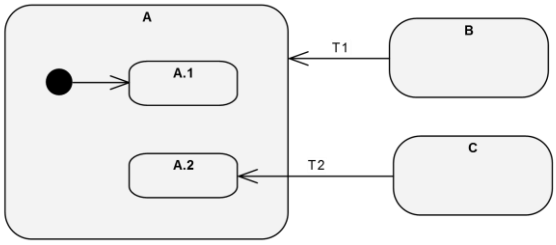

Abbildung 64: Betreten von zusammengesetzten Zuständen

Semantik beim Betreten von zusammengesetzten Zuständen

- *Default Entry (Trigger T1):* Beim Betreten des Zustands A aus Zustand B heraus wird Startzustand durchlaufen und dann der Zustand A.1 eingenommen.
- *Explicit Entry (Trigger T2):* Wird der Zustand A vom Zustand C beginnend betreten, wird der Startknoten nicht durchlaufen und unmittelbar der Zustand A.2 eingenommen.

Eine weitere Möglichkeit zum Betreten bietet das Modellierungskonstrukt der Historie.

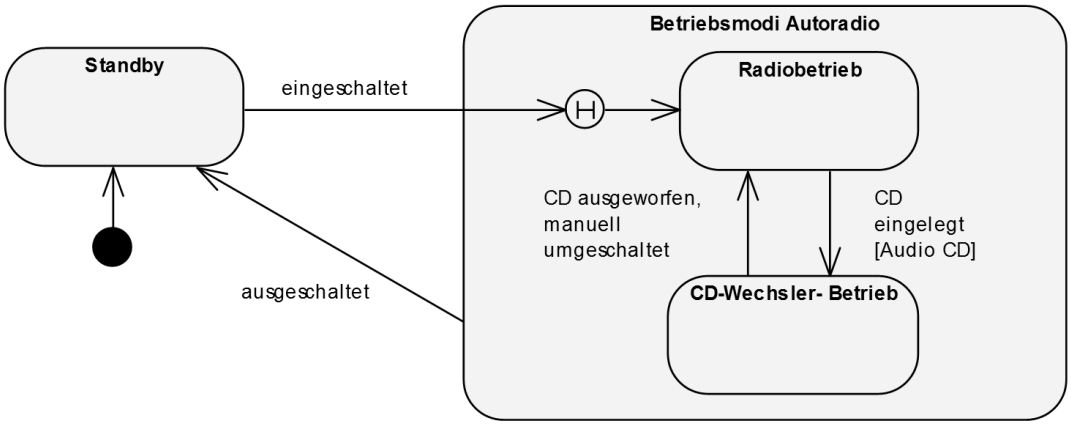

Abbildung 65: Flache Historie

Falls der Zustand „Betriebsmodi Autoradio“ betreten wird, wird der Zustand aktiv, der bei vorherigem Verlassen von „Betriebsmodi Autoradio“ aktiv war. Nur in dem Sonderfall des erstmaligen Betretens wird die Transition von der Historie aus durchlaufen, und der Zustand „Radiobetrieb“ wird aktiv. In der Abbildung ist die sogenannte „Flache Historie“ dargestellt. Besteht eine tiefere Hierarchie von zusammengesetzten Zuständen, kann die „tiefe Historie“ verwendet werden, die dann nicht nur den Unterzustand auf oberer Ebene identifiziert, sondern dafür sorgt, dass das Identifizieren des zuletzt eingenommenen Zustandes bis auf die Blattebene der Hierarchie propagiert wird. Diese tiefe Historie wird mit H* notiert.

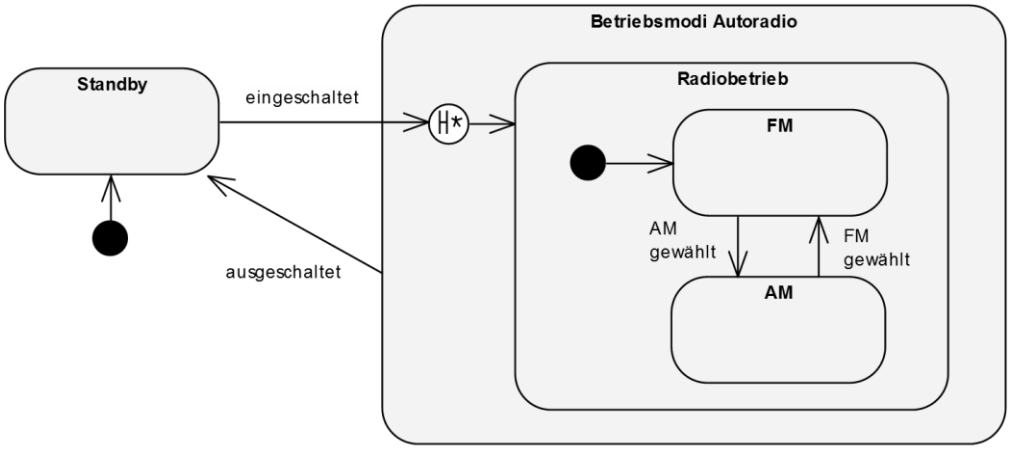

Abbildung 66: Tiefe Historie

4.4.4.5.3 Verlassen zusammengesetzter Zustände

Für das Verlassen werden auch verschiedene Möglichkeiten unterschieden.

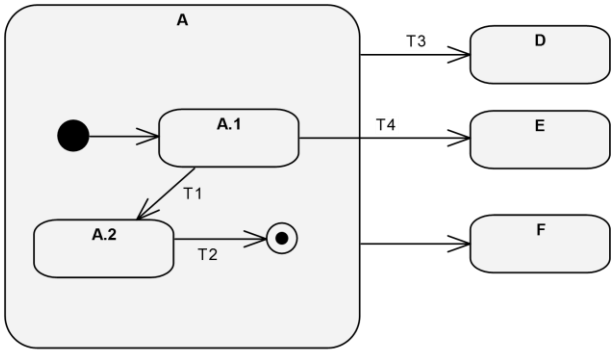

Abbildung 67: Verlassen von zusammengesetzten Zuständen

Verlassen eines zusammengesetzten Zustandes

- Erreichen des Endzustands (Trigger T2): Hier muss eine triggerlose Transition existieren, die dann durchlaufen wird. Der aktive Zustand ist danach F.
- Transition eines Unterzustandes (Trigger T4): Dies entspricht der logischen Semantik: Falls A.1 aktiv ist, und das Signal T4 empfangen wird, wird der Zustand E aktiv.
- Transition des zusammengesetzten Zustands (Trigger T3): hier zeigt sich die Stärke dieses Modellierungskonstrukts: Unabhängig davon, welcher Unterzustand gerade aktiv ist (A.1 oder A.2), wird dieser verlassen, sobald der Trigger T3 vorliegt. Mit dieser Art wird auch der Gedanke der Abstraktion unterstützt, da das Verhalten auf der oberen Ebene vollständig unabhängig von der Situation innerhalb von A definiert wird.

4.4.4.5.4 Finden von zusammengesetzten Zuständen

Diese Art des Auffindens von Zuständen erscheint zunächst einfach mit der folgenden Regel: Falls das System in mehreren Zuständen ein ähnliches Verhalten aufweisen soll (Verlassen des Zustands, Aufruf von Funktionen) können diese zu einem zusammengesetzten Zustand zusammengefasst werden. Jedoch kann es sein, dass durch diese Betrachtung ein Zustand in mehreren zusammengesetzten Zuständen vorkommen kann. Dies darf jedoch nicht der Fall sein. Hier müssen Sie dann aufgrund der Wichtigkeit für die betrachtete Fachlichkeit einen komplexen Zustand als Vater des Zustands wählen.

Im Allgemeinen ergeben sich die zusammengesetzten Zustände jedoch relativ natürlich aus der Fachlichkeit des Betrachtungsgegenstands. Zum Beispiel besitzt ein Zimmerventilator auf oberer Ebene die beiden Zustände „An“ und „Aus“, der Zustand „An“ kann dann je nach gewählter Stufe weiter unterteilt werden (Langsam, Schnell).

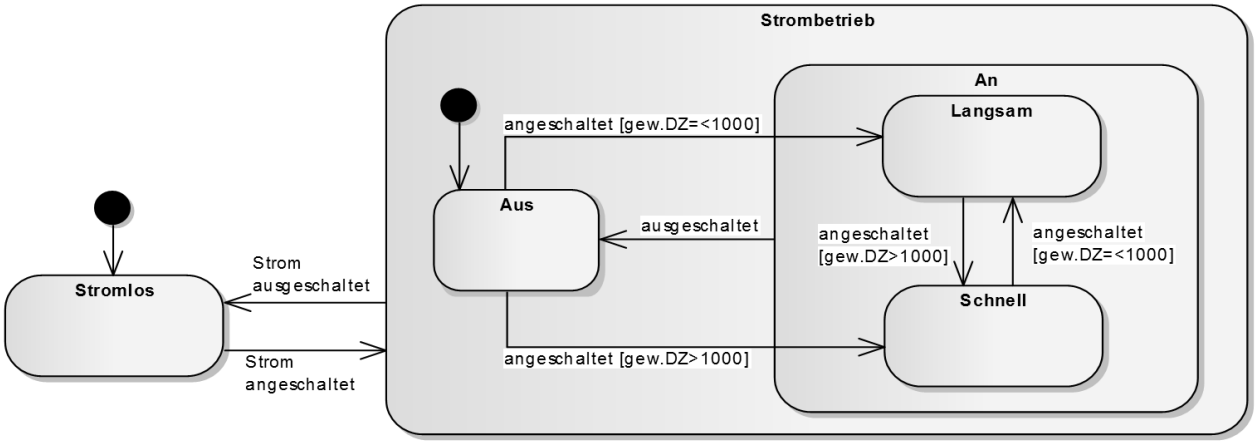

Abbildung 68: Zustände eines Ventilators

4.4.4.6 Unterzustandsautomatenzustand

4.4.4.6.1 Syntax und Semantik

Ein Unterzustandsautomatenzustand wird wie ein einfacher Zustand dargestellt. Jedoch existieren zwei mögliche Erweiterungen gegenüber einem einfachen Zustand. Die Bezeichnung des Unterzustandsautomatenzustands und die Bezeichnung des mit diesem Zustand verknüpften Zustandsautomaten werden durch einen Doppelpunkt separiert, oder es wird eine Brille unten rechts dargestellt.

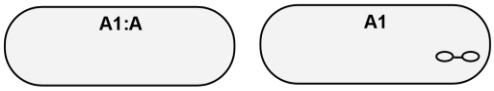

Abbildung 69: Syntax eines Unterzustandsautomatenzustands

Mit der Einführung des Unterzustandsautomatenzustands wird die Idee der Hierarchisierung, wie sie durch zusammengesetzte Zustände eingeführt wurde, weiter fortgesetzt. Die untergeordneten Zustände eines zusammengesetzten Zustands werden in einen eigenen Zustandsautomaten ausgelagert (in ein eigenes Diagramm). Dieser Automat wird dann auf einer höheren Ebene durch den Unterzustandsautomatenzustand referenziert.

Um die im Abschnitt 4.4.4.5 eingeführten Möglichkeiten auch in einem Unterzustandsautomaten weiterhin nutzen zu können, werden in der UML die Eintritts- und Austrittspunkte (Entry- und Exit-Points) eingeführt. Mit diesen Modellierungskonstrukten kann sowohl ein Explicit Entry als auch eine Transition von einem Unterzustand zum Verlassen modelliert werden. Dies führt den Gedanken der Abstraktion fort, wie er in Abschnitt 4.4.4.5 eingeführt wurde.

Abbildung 70 zeigt den Übergang von einem zusammengesetzten Zustand zu der Auslagerung in einen Unterzustandsautomatenzustand.

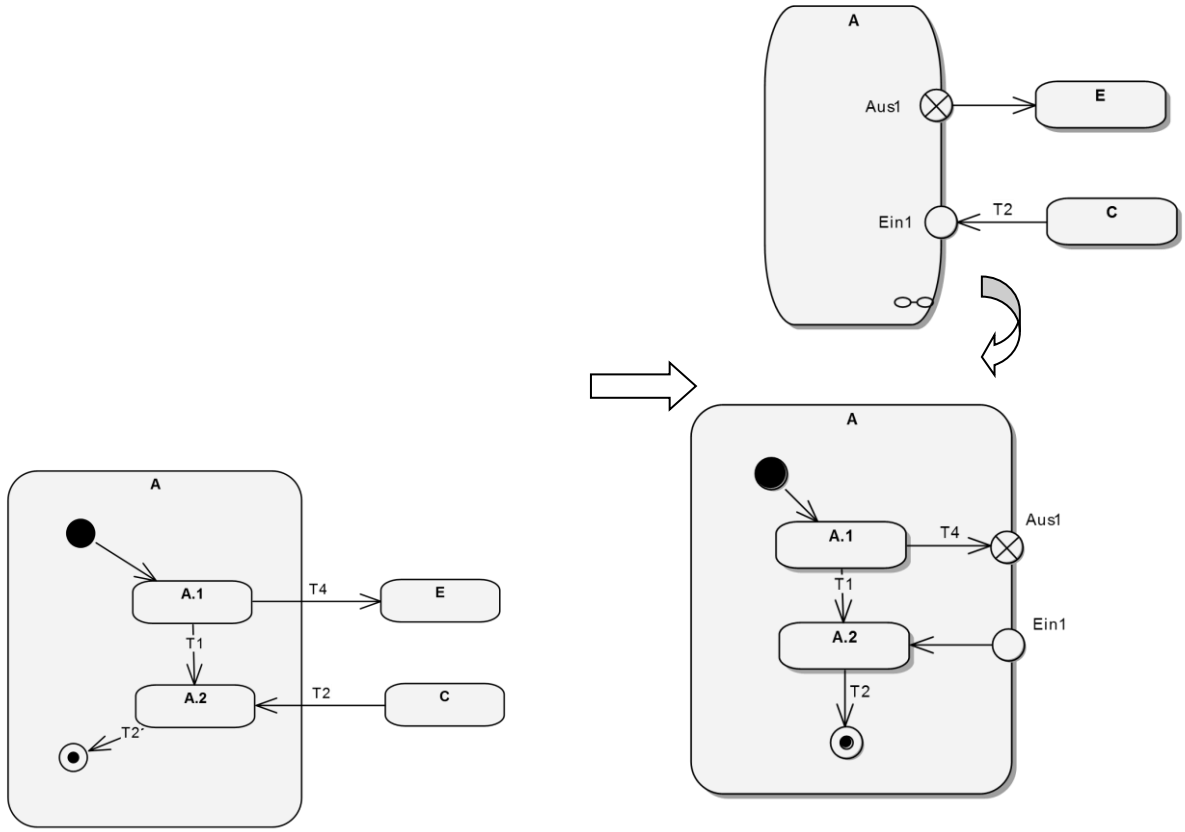

Abbildung 70: Verwendung von Ein- und Austrittspunkten

Im linken Teil ist ein zusammengesetzter Zustand modelliert, im rechten Teil die Verwendung eines Unterzustandsautomatenzustands. Beachten Sie dabei, wo die Trigger T4 und T2 in der rechten Lösung notiert werden. Ein Beispiel für die Verteilung von Guards ist in dem Beispiel im folgenden Abschnitt gegeben.

4.4.4.6.2 Finden von Unterzustandsautomaten

Zur Identifikation von Unterzustandsautomaten kann die gleiche Heuristik angewendet werden, wie zur Identifikation zusammengesetzter Zustände (siehe Abschnitt 4.4.2). Zusätzlich sollte darauf geachtet werden, dass in einem Unterzustandsautomaten mehrere abstraktere Zustandsautomaten verwenden werden können und dass durch die Verwendung solcher Konzepte die Diagramme übersichtlicher gestaltet werden können.

Als Beispiel für eine solche Reduktion der Komplexität in einem Zustandsautomaten dient das Beispiel eines Zimmerventilators mit dem abstrakten Zustandsautomat und der Verfeinerung des Zustands *An*.

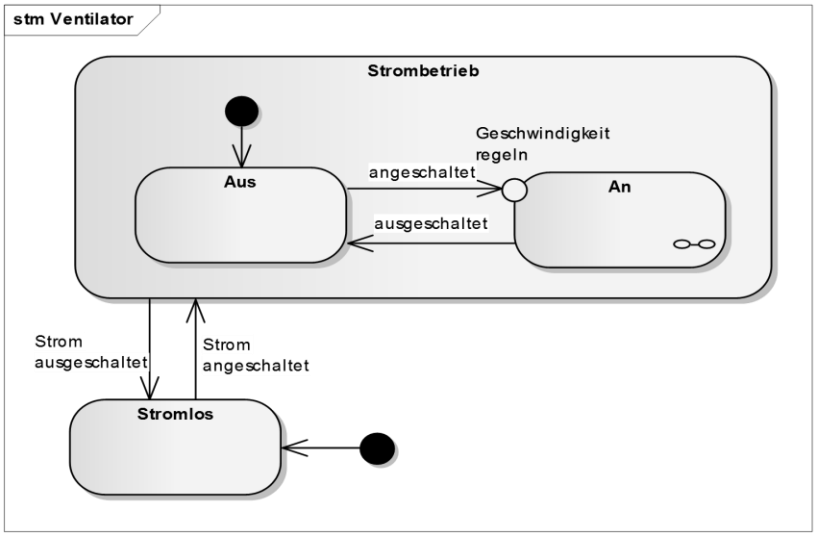

Abbildung 71: Zustandsautomat eines Ventilators

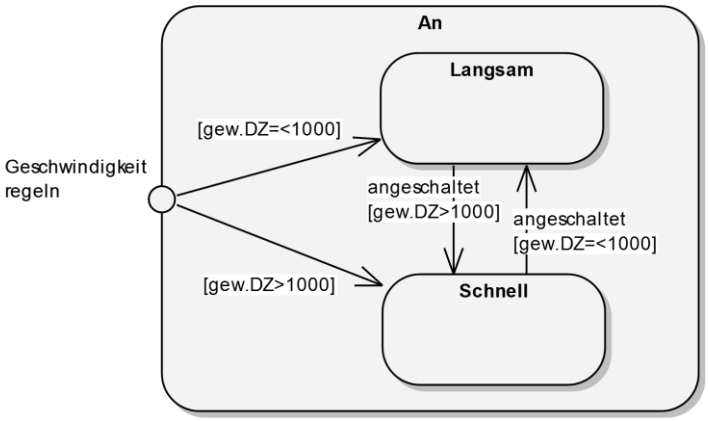

Abbildung 72: Hierarchische Zustände eines Ventilators

4.4.4.7 Orthogonale Regionen

Mit den orthogonalen Regionen definiert man zwei oder mehr Teile eines Zustandsautomaten, die unabhängig voneinander auf Ereignisse reagieren können.

4.4.4.7.1 Syntax und Semantik

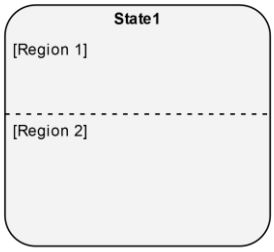

Abbildung 73: Syntax von orthogonalen Regionen

Ein Zustand kann in mehrere, zueinander orthogonale Regionen zerlegt werden. In jeder Region kann man einen eigenen Zustandsautomaten, ähnlich den zusammengesetzten Zuständen, modellieren. Damit haben Sie die Möglichkeit, die Anzahl der Zustände zu verringern, wenn Sie die Zustände in mehrere, voneinander unabhängige Mengen verteilen können.

Als Erklärung dient das folgende Beispiel eines Infotainmentsystems, das sowohl ein Radio als auch ein Navigationsystem bietet (siehe Abbildung 74). Nach dem Einschalten können unabhängig voneinander das Radio und die Navigation auf Standby geschaltet werden. Weiterhin kann sich die Navigation in der Zieleingabe oder der Routenführung befinden. Unabhängig davon kann sich das Radio im Radiobetrieb oder im CD-Wechslerbetrieb befinden. Aus diesen sechs möglichen Zuständen für die beiden Teile würden sich insgesamt neun Unterzustände (3 mal 3) des Zustands *Aktiv* (nach dem Einschalten) ergeben. Da jeweils drei Zustände voneinander unabhängig sind, lässt sich der Zustand *Aktiv* in zwei orthogonale Regionen aufteilen, und es ergibt sich der folgende Zustandsautomat.

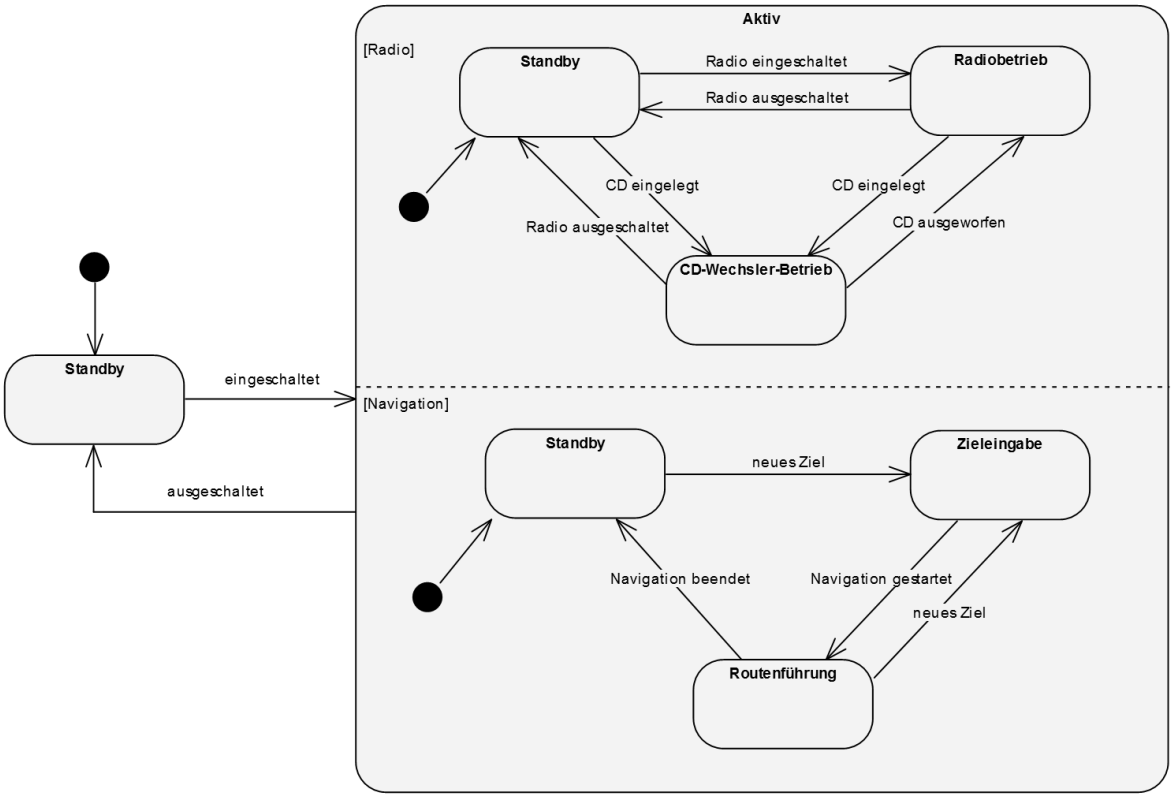

Abbildung 74: Orthogonale Regionen eines Infotainmentsystems

Für die Unabhängigkeit der Zustände muss gelten:

- Das Verhalten in einer Region hängt nicht von dem aktuellen Zustand in der anderen Region ab.
- Es existieren keine Transitionen über die Grenzen der Regionen hinweg.

Zu bemerken ist, dass auch bei der Verwendung der orthogonalen Regionen das Paradigma aus Abschnitt 4.4.2 nicht verletzt wird. Das System befindet sich zu jedem Zeitpunkt noch immer in genau einem Zustand, der sich allerdings aus der Kombination der jeweils aktiven Zustände der einzelnen Regionen ergibt.

In dem obigen Beispiel ist für das Verlassen des aktiven Zustands eine Möglichkeit wie bei den zusammengesetzten Zuständen verwendet worden. Für das Betreten ist das Modellierungskonstrukt der Parallelisierung verwendet, um auszudrücken, welche beiden Unterzustände gleichzeitig eingenommen werden sollen. Neben diesen Möglichkeiten existiert noch eine Vielzahl weiterer Ein- und Austrittsmöglichkeiten. Für eine vollständige Betrachtung wird auf [RuQu2012] verwiesen.

Die folgende Abbildung zeigt einen Zustandsautomaten, bei dem die orthogonalen Regionen des Zustandsautomaten aus Abbildung 74 aufgelöst wurden. Man erkennt, dass aus den dort modellierten sechs Zuständen nun neun Zustände abgeleitet werden mussten. Die Anzahl der Transitionen steigt dabei in noch größerem Maße.

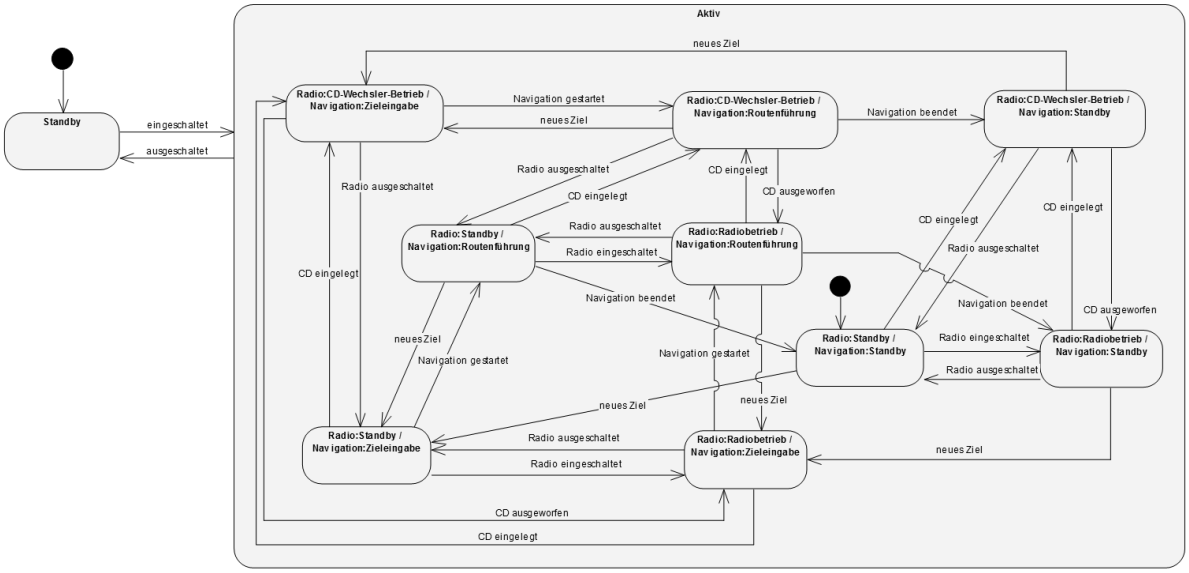

Abbildung 75: Aufgelöste orthogonale Regionen

4.4.4.7.2 Finden von orthogonalen Regionen

Die Regionen zu finden bzw. zu erkennen, dass orthogonale Regionen gebildet werden können, ist nicht immer einfach. Hier hat es sich bewährt, die Modellierung zunächst ohne dieses Modellierungskonstrukt zu starten. Falls die Automaten unübersichtlich werden, können Sie sich dann Gedanken machen, ob nicht vielleicht die Benennung der Zustände auf solche Regionen hindeuten. Häufig ist auch ein Indiz für solche Regionen, dass Teile eines Zustandes (dessen Verfeinerung) in mehreren, voneinander unabhängigen Teilen diskutiert wird.

4.4.5 Typische Zustandsautomaten / Modellierungsszenarien

4.4.5.1 Generischer Zustandsautomaten für technische Systeme

In der folgenden Abbildung ist ein Zustandsautomat angegeben, der als Vorlage für einen Zustandsautomaten eines technischen Systems dienen kann.

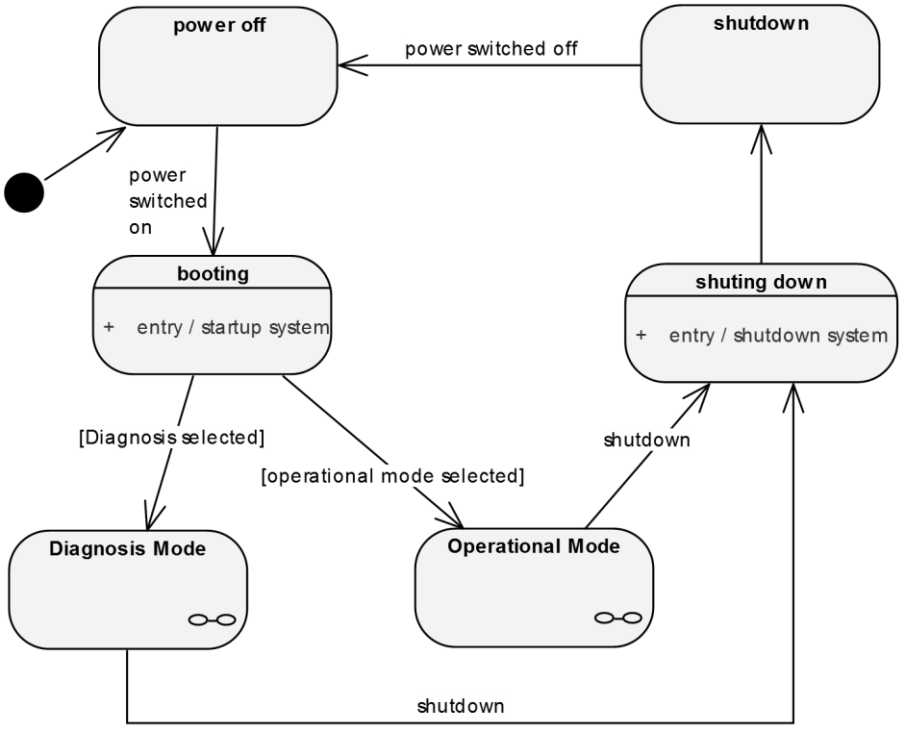

Abbildung 76: Generischer Zustandsautomat eines technischen Systems

In diesem Automaten sollten die beiden Zustände „Diagnosis Mode“ und „Operational Mode“ weiter verfeinert werden. Diese Verfeinerungen sind jedoch sehr von dem betrachteten System abhängig, weshalb an dieser Stelle keine weiteren Aussagen über die Gestalt dieser Zustände getroffen werden können.

Weiterhin können in diesen Zustandsautomaten bei Bedarf weitere Zustände integriert werden. In Infotainmentsystemen in der Automobilbranche wird zum Beispiel ein „Transport mode“ definiert, in dem das System keine Eingaben annimmt. Dieser wäre parallel zu dem „Diagnose Mode“ und „Operational Mode“ zu sehen.

4.4.5.2 Zustände von Objekten eines geschäftsorientierten Systems

Als typisches Beispiel für die Zustände eines Objekts in einem geschäftsorientierten System soll das Objekt *Urlaubsantrag* dienen. Dessen Zustandsautomat ist in Abbildung 77 gezeigt, wobei auf die vollständige Definition von Triggern, Guards und Funktionen verzichtet wurde:

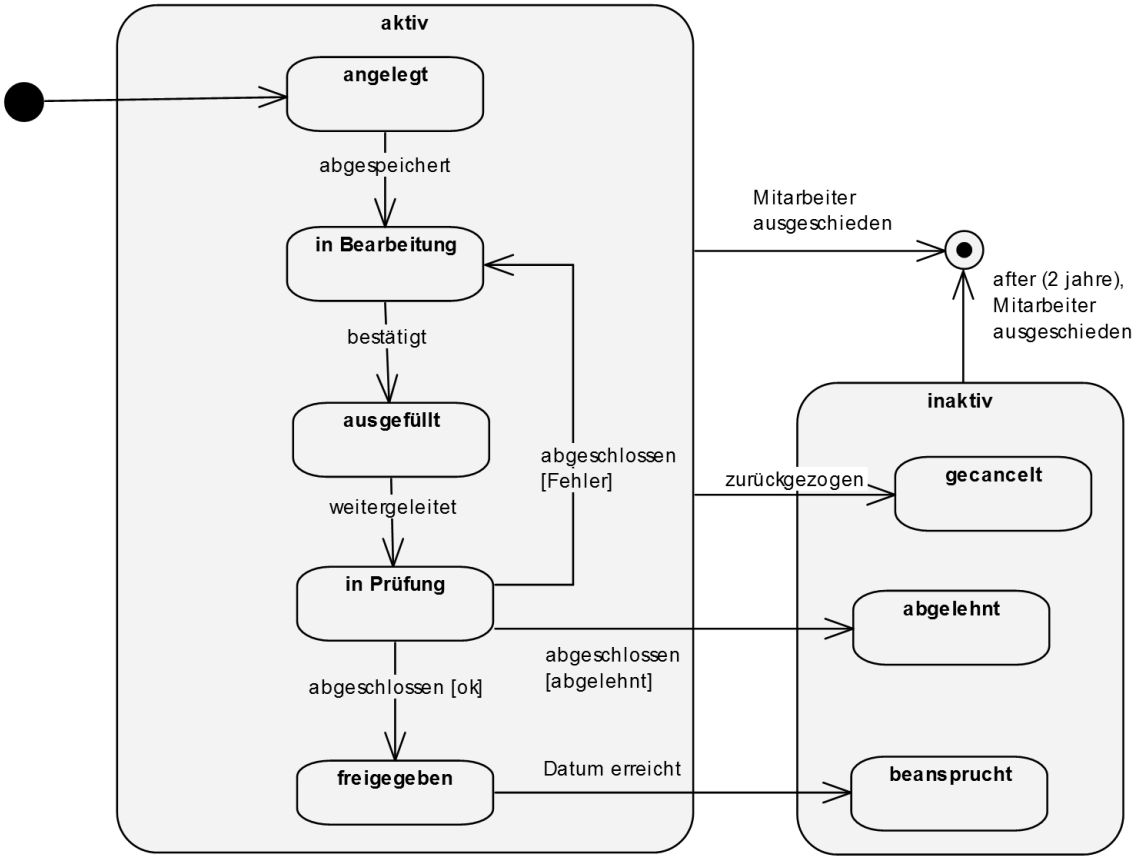

Abbildung 77: Zustände eines Urlaubsantrages

Wie man in dem Automaten sieht, korrespondieren die Zustände des Objekts zu den Zeitintervallen, in denen der Urlaubsantrag einen Stand hat, der es wert ist, persistiert zu werden. Dies entspricht auch den möglichen Zeitpunkten der Bearbeitung durch Use-Cases, da dieser eine in sich abgeschlossene Interaktion zwischen Akteur und System beinhaltet. Dies bedeutet in der Folge, dass die Stände des Urlaubsantrags nach dem Ablauf eines Use-Cases gesichert werden müssen. Technisch gesprochen bedeutet dies, dass diese Information mit in der Datenbank abgelegt werden muss, damit die fachlogische Implementierung weiß, welche Schritte für einen bestimmten Urlaubsantrag erlaubt sind.

Der enge Zusammenhang zwischen den Zuständen und den Use-Cases zur Bearbeitung eines solchen Antrags lässt sich auch anders ausdrücken: Die Zustände des Objekts geben die Nachbedingungen an, die bei einem Use-Case definiert wurden.

4.5 Vertiefende Literatur

Datenflusssicht

- DeMarco, Tom: Structured Analysis and System Specification, Yourdon Press, Prentice Hall, 1979

Kontrollflusssicht, insbesondere Aktivitätsdiagramme

- Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, 2004
- Rupp, C.; Queins, S.: UML 2 Glasklar – Praxiswissen für die UML-Modellierung. Carl Hanser Verlag 2012.

Use-Case-Modellierung und –Spezifikation:

- Jacobson, I.; Christerson, M.; Jonsson, P.; Oevergaard, G.: Object Oriented Software Engineering – A Use-Case Driven Approach. Addison-Wesley, Reading, 1992.
- Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, 2004.
- Cockburn, Alistair: Writing Effective Use-Cases, Addision Wesley, 2001 (auch in deutsch erhältlich: Use-Cases effektiv erstellen, mitp, 2003)

Zustandssicht

- Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, 2004
- Rupp, C.; Queins, S.: UML 2 Glasklar – Praxiswissen für die UML-Modellierung. Carl Hanser Verlag 2012.

5 Szenariomodellierung

Szenarien sind heutzutage ein wesentliches Mittel im Requirements Engineering, um z.B. die Systemvision und die Ziele von Stakeholdern zu konkretisieren oder um die Mehrwerterbringung für die Nutzer des Systems zu beschreiben. Hierbei haben Szenarien den Charakter von Beispielen, die die Nutzung des zu entwickelnden Systems durch Menschen oder andere Systeme betrachten (vgl. z.B. [Caro1995]). Neben dem Einsatz von Szenarien zur exemplarischen Beschreibung der Nutzung des betrachteten Systems, können Szenarien auch zur präzisen Spezifikation funktionaler Anforderungen eingesetzt werden. In diesem Falle werden in der zugehörigen Szenariosicht alle in der Systemnutzung auftretenden Szenarien in hoher Präzision, z.B. durch Sequenzdiagramme der UML oder Message Sequence Charts nach ITU-Standard [ITU2004], spezifiziert.

5.1 Zweck

Seit den frühen 1990er Jahren werden im Requirements Engineering vermehrt Szenarien eingesetzt, um die systematische Spezifikation von Anforderungen zu unterstützen (vgl. z.B. [Pott1995]). Eine wesentliche Erkenntnis, die in dem Einsatz von Szenarien mündete, war, dass wenn der Ausgangspunkt für das Requirements Engineering die grobe Systemvision bzw. die Ziele der Stakeholder sind, es in vielen Fällen schwierig ist, darauf basierend unmittelbar die Anforderungen an das System vollständig und korrekt zu spezifizieren (vgl. z.B. [DaLF1993]). Szenarien fokussieren auf die interaktionsbasierte Sicht, eine spezielle Verhaltenssicht auf die funktionalen Anforderungen des Systems, bei der das Verhalten des Systems durch zusammengehörende Sequenzen von Interaktionen zwischen Kommunikationspartnern betrachtet wird. Im Mittelpunkt der interaktionsbasierten Sicht stehen die Kommunikationspartner, die entweder Systeme bzw. Personen im Systemkontext oder das zu entwickelnde System repräsentieren sowie die Interaktionen zwischen diesen. Eine **Interaktion** zwischen Kommunikationspartnern ist dabei eine Folge ausgetauschter Nachrichten zwischen diesen. Diese Nachrichten können dabei Informationen bzw. Daten sein, die über Kommunikationskanäle zwischen den kommunizierenden Akteuren ausgetauscht werden. Im Requirements Engineering werden in Interaktionen darüber hinaus aber auch Nachrichten in Form materieller Flüsse zwischen Kommunikationspartnern betrachtet, wie z.B. ein Materialfluss oder ein Geldfluss zwischen Kommunikationspartnern.

Ein **Szenario** ist eine Interaktion zwischen Kommunikationspartnern (häufig zwischen dem betrachteten System und Akteuren im Systemkontext), die zu einem gewünschten (oder ggf. ungewünschten) Resultat führt. Die Szenariomodellierung wird häufig eingesetzt, um die Systemvision und die Ziele der Stakeholder im Hinblick auf die gewünschte Nutzung des Systems zu konkretisieren. Die Szenariomodellierung beschränkt sich dabei i.d.R. nicht nur auf die Schnittstelle des betrachteten Systems in Form des unmittelbaren Nachrichtenaustausches zwischen Akteuren und dem System, sondern betrachtet auch Nachrichten, die zwischen Akteuren im Systemkontext ausgetauscht werden. Bei der Szenariomodellierung werden damit nicht nur die Anforderungen an das betrachtete System modelliert, sondern auch der Interaktionskontext von Nachrichten, die zwischen Akteuren und dem betrachteten System ausgetauscht werden.

Im Requirements Engineering wird häufig der Mehrwert für einen Akteur im Systemkontext als wesentliches Resultat eines Szenarios gesehen. Das folgende Beispiel zeigt ein einfaches,

in natürlicher Sprache beschriebenes Szenario, welches eine Interaktion dokumentiert, die zwischen einer Person (Karl) und einem Online-Shop abläuft, damit Karl einen Einkauf tätigen kann.

Beispiel: Szenario „Einkauf im Online-Shop“

Karl wählt im Produktkatalog des Online-Shops die gewünschten Produkte aus und bestätigt dann, dass er den Kauf abschließen möchte. Der Online-Shop zeigt Karl die von ihm ausgewählten Produkte mit Anzahl und Kaufpreis sowie den Gesamtrechnungsbetrag an. Der Online-Shop erbittet von Karl die Bestätigung der gewünschten Einkäufe. Nachdem Karl die gewünschten Einkäufe bestätigt hat, erfragt der Online-Shop die Lieferadresse für die Sendung. Karl gibt die gewünschte Lieferadresse ein und bestätigt dies. Nach Bestätigung der Lieferadresse erfragt der Online-Shop die Zahlungsdaten von Karl. Karl gibt die Zahlungsdaten ein und bestätigt diese. Anschließend zeigt der Online-Shop die Gesamtbestellung sowie die gewünschte Lieferadresse und die Zahlungsdaten an und bittet Karl, diese Bestellung zu bestätigen. Karl bestätigt die Bestellung, woraufhin der Online-Shop eine Auftragsbestätigung anzeigt.

Der zugehörige Mehrwert, den der Akteur (Karl) durch die Nutzung des Online-Shops erfährt, besteht darin, dass Karl durch das Aufgeben der Bestellung auf elektronischem Wege die gewünschten Produkte einkaufen kann.

5.2 Zusammenhang zwischen Szenarien und Use-Cases

Im Requirements Engineering werden verschiedene Arten von Szenarien unterschieden. Eine umfangreiche Betrachtung der unterschiedlichen Typen von Szenarien findet sich z.B. in [RAC+1998]. Im Folgenden werden zwei häufig in der Praxis vorzufindende Differenzierungen von Szenarien und die zugehörigen Arten vorgestellt.

Eine häufig auftretende Differenzierung von Szenarien unterscheidet zwischen Hauptszenarien, Alternativszenarien und Ausnahmeszenarien. Diese Unterscheidung ist ein zentrales Element von Use-Case-basierten Ansätzen (wie z.B. [JCJO1992]), bei denen Szenarien, die sich auf einen spezifischen Mehrwert beziehen, innerhalb eines Use-Cases gruppiert und ergänzend zueinander dokumentiert sind (siehe Abschnitt 4.2.5). Die Verwendung von Haupt-, Alternativ- und Ausnahmeszenarien ist aber nicht notwendigerweise auf Use-Case-basierte Ansätze beschränkt. Ein **Hauptszenario** ist ein Szenario, welches, bezogen auf ein spezifisches Resultat (z.B. einen spezifischen Mehrwert), eine überwiegend auftretende Sequenz von Interaktionen zur Erreichung dieses Resultates beschreibt. Ein **Alternativszenario** ist ein Szenario, welches, bezogen auf ein spezifisches Resultat (z.B. einen spezifischen Mehrwert), das in Bezug auf ein Hauptszenario eine alternative Sequenz von Interaktionen zur Erreichung des Resultates beschreibt. Ein **Ausnahmeszenario** ist ein Szenario, welches eine Sequenz von Interaktionen beschreibt, die durchlaufen werden müssen, wenn ein definiertes Ausnahmeereignis eintritt. Im Requirements Engineering werden Ausnahmeszenarien zur kontrollierten Behandlung von Ausnahmesituationen im Betrieb häufig ergänzend zu Hauptszenarien und Alternativszenarien spezifiziert. Typischerweise ist in der Praxis die Anzahl von Ausnahmesezenarien in den meisten Fällen erheblich größer als die Zahl der Alternativszenarien eines Hauptszenarios, da mit Hilfe der Ausnahmeszenarien (und zugehöriger Ausnahmenereignisse) möglichst alle Situationen erfasst werden sollten, die bei der Ausführung der Haupt- oder Alternativszenarien auftreten können und dazu führen, dass die entsprechenden Szenarien (bzw. der zugehörige Use-Case, siehe Abschnitt 4.2) im Betrieb des Systems nicht mehr erfolgreich ausgeführt werden können. Die jeweiligen Ausnahmes-

zenarien spezifizieren jeweils eine kontrollierte Ausnahmebehandlung zu einem definierten Ausnahmeereignis.

5.3 Ansätze zur Szenariomodellierung

Die Modellierung von Szenarien gestattet es, umfangreiche und komplexe Sachverhalte, die das interaktionsbasierte Verhalten des Systems betreffen, auf gut verständliche und strukturierte Weise zu dokumentieren. Zur Modellierung von Szenarien eignen sich dabei solche Diagrammtypen, die es erlauben, eine Ordnung von Interaktionen zwischen Kommunikationspartnern in visueller Form zu dokumentieren. Heutzutage werden zur Modellierung von Szenarien häufig Sequenzdiagramme eingesetzt. Bei Sequenzdiagrammen werden die verschiedenen, an der zu modellierenden Interaktionssequenz beteiligten Kommunikationspartner in der horizontalen Dimension angeordnet. Die Interaktionen zwischen den Kommunikationspartnern werden in der Reihenfolge ihres Auftretens in der vertikalen Dimension modelliert. Auf diese Art und Weise können z.B. auch Szenarien von Use-Cases durch Diagramme genauer spezifiziert werden (vgl. Abschnitt 4.2).

In der Telekommunikationsbranche werden etwa Message Sequence Charts (MSCs) der International Telecommunication Union (ITU) nach dem Standard ITU-T Z.120 verwendet [ITU2004], die durch den hohen Formalisierungsgrad weitreichende Möglichkeiten zur maschinellen Verarbeitung, z.B. hinsichtlich Qualitätsprüfung (z.B. auf Widerspruchsfreiheit und Vollständigkeit) oder in generativen Entwicklungsansätzen bieten. Durch die Verwendung sogenannter h(high-level)MSCS (ähnlich den Interaktionsübersichtsdiagrammen in UML 2), können umfangreiche und komplexe Modelle der Szenariosicht geeignet strukturiert werden. Der ITU-T Z.120 Standard existiert seit 1992, wurde seitdem kontinuierlich weiterentwickelt und hat insbesondere die **Sequenzdiagramme der UML** [OMG2010c, OMG2010b] und **Sequenzdiagramme der SysML** [OMG2010a] in wesentlichen Teilen geprägt. Die Verwendung von Sequenzdiagrammen der UML/SysML hat den Vorteil, dass die UML/SysML in der Praxis deutlich verbreiteter ist als konkurrierende Modellierungsansätze, wie z.B. solche der ITU. Darüber hinaus sind die in UML/SysML Sequenzdiagrammen modellierten Szenarien und zugehörige Modellelemente durch das Metamodell der UML bzw. SysML mit anderen Sichten der Anforderungsmodellierung integrierbar, wenn hierzu ebenfalls auf Diagrammtypen der UML bzw. SysML zurückgegriffen wird.

Neben den originären Sequenzdiagrammen der UML und SysML bietet die UML mit den **Kommunikationsdiagrammen** noch einen weiteren Diagrammtyp, mit dessen Hilfe ebenfalls Szenarien modelliert werden können. Im Vergleich zu den Sequenzdiagrammen, die in erster Linie auf die Reihenfolge der Interaktionen zwischen Kommunikationspartnern fokussieren, legen Kommunikationsdiagramme der UML den Schwerpunkt auf die Visualisierung der bilateralen Interaktionen zwischen Kommunikationspartnern. Die Reihenfolge der Interaktionen wird hierbei durch Sequenznummern an den Interaktionen angegeben.

5.4 Einfache Beispiele für ein modelliertes Szenario

Abbildung 78 zeigt die Modellierung eines einfachen Szenarios in Form eines Sequenzdiagrammes der UML (a) und eines Kommunikationsdiagrammes der UML (b).

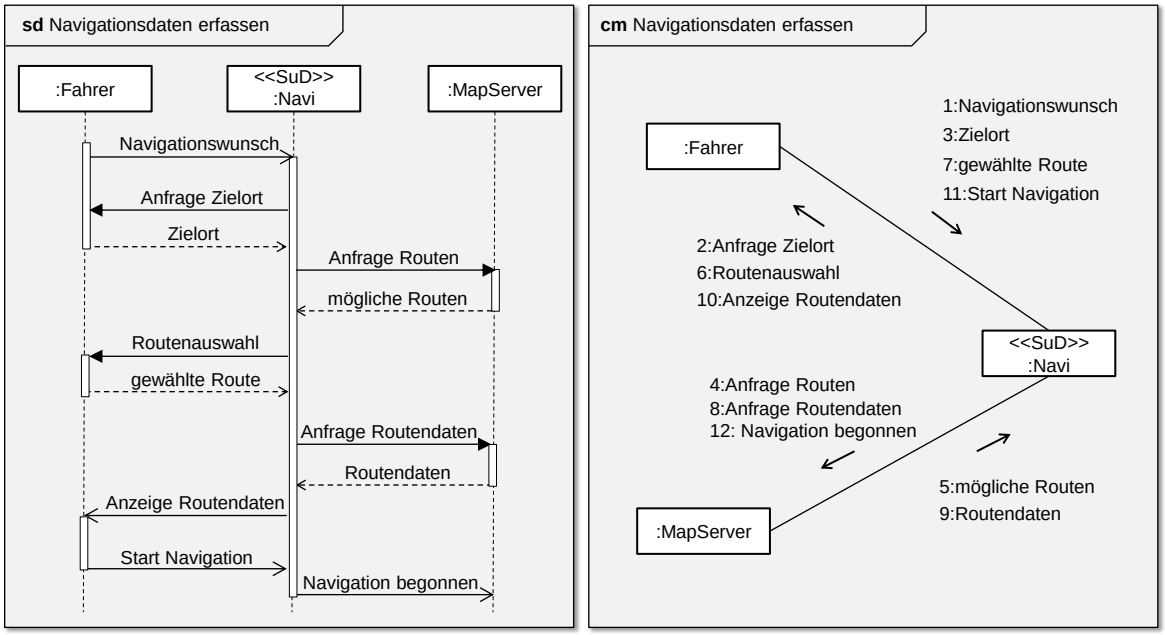

Abbildung 78: Modellierung eines Szenarios durch (a) Sequenzdiagramm; (b) Kommunikationsdiagramm

Beide Diagramme modellieren das Szenario „Navigationsdaten erfassen“. Der Name des Szenarios ist im oberen Bereich des Rahmens angegeben. Die verwendeten Schlüsselwörter „sd“ und „cm“ benennen jeweils den Typ des Diagramms, durch das das entsprechende Szenario modelliert wurde. In Abbildung 78 steht „sd“ für Sequenzdiagramm und „cm“ für Kommunikationsdiagramm.[12]

Das Sequenzdiagramm im linken Bereich von Abbildung 78 zeigt eine Folge von Interaktionen zwischen Instanzen der Kommunikationspartner „:Fahrer“, „:Navi“ und „:MapServer“, die ablaufen muss, damit der Fahrer am Navigationsgerät die Navigationsdaten erfassen kann. Der Betrachtungsgegenstand ist im Sequenzdiagramm durch den Stereotyp <<SuD>> (für: System-under-Development) markiert, um im Diagramm visuell die Trennung zwischen dem Betrachtungsgegenstand und den Akteuren im Systemkontext deutlich zu machen. Wie aus dem Diagramm ersichtlich, wird bei Sequenzdiagrammen die Reihenfolge der Interaktionen in der vertikalen Dimension modelliert; in der horizontalen Dimension werden die verschiedenen, an dem jeweiligen Szenario beteiligten Instanzen von Kommunikationspartnern, aufgeführt, wobei der „:“ vor dem Namen des Kommunikationspartners anzeigt, dass jeweils eine konkrete Instanz betrachtet wird. Die Pfeilspitze zeigt jeweils an, in welche Richtung der Nachrichtenaustausch stattfindet.

Das Kommunikationsdiagramm im rechten Bereich von Abbildung 78 repräsentiert ebenfalls das Szenario „Navigationsdaten erfassen“. Bei diesem Diagramm wird allerdings der zeitliche Ablauf der Interaktionen nicht in der vertikalen Dimension dokumentiert, sondern ist durch die Angabe von Sequenznummern an den Interaktionen annotiert. Das Kommunikationsdiagramm visualisiert durch eine Linie zwischen Kommunikationspartnern die Existenz einer unmittelbaren Kommunikationsbeziehung. Die über diese Kommunikationsbeziehung auftretenden Interaktionen werden durch Nachrichten dokumentiert. Jede dieser Nachrichten wird durch einen Namen, die zugehörige Sequenznummer der Nachricht im Szenario sowie durch die Richtung des Nachrichtenflusses angegeben.

[12] Da Kommunikationsdiagramme eine alternative Form der Darstellung für „einfache“ Sequenzdiagramme sind, wird teilweise bei Kommunikationsdiagrammen ebenfalls das Schlüsselwort „sd“ verwendet

Bei Kommunikationsdiagrammen werden die Kommunikationsbeziehungen zwischen jeweils zwei Kommunikationspartnern in den Vordergrund der Visualisierung gestellt. Demgegenüber werden die zeitlich bzw. logische Abfolge der Interaktionen von Szenarien besser durch Sequenzdiagramme visualisiert. Aufgrund der unterschiedlichen Schwerpunkte der Visualisierung muss der Requirements Engineer situationsabhängig entscheiden, welcher der beiden Diagrammtypen für den jeweiligen Verwendungszweck geeigneter ist (↑ Pragmatische Qualität). Falls unterschiedliche Verwendungszwecke es notwendig machen, kann ein Szenario in beiden Diagrammtypen modelliert werden, oder das Sequenzdiagramm bzw. das Kommunikationsdiagramm kann aus dem Diagramm des jeweils anderen Diagrammtyps maschinell konstruiert werden, wobei wesentlich ist, dass komplexe Interaktionen (z.B. bei der bedingten Wiederholung von Nachrichten oder alternativen Nachrichten) nicht oder nur sehr schwer durch Kommunikationsdiagramme dargestellt werden können.

Im Folgenden werden die verschiedenen Modellierungskonstrukte zur Szenariomodellierung mit Sequenzdiagrammen der UML/SysML bzw. mit Kommunikationsdiagrammen der UML vorgestellt und deren spezifische Bedeutung für die Modellierung von Anforderungen erläutert. Weiterführende Informationen zu den Modellierungskonstrukten von Sequenzdiagrammen und Kommunikationsdiagrammen finden sich in [OMG2010b] oder [OMG2010a].

5.5 Szenariomodellierung mit Sequenzdiagrammen

Abbildung 79 zeigt die Modellierungskonstrukte der OMG UML/SysML für Sequenzdiagramme, die zur Modellierung von Szenarien verwendet werden.

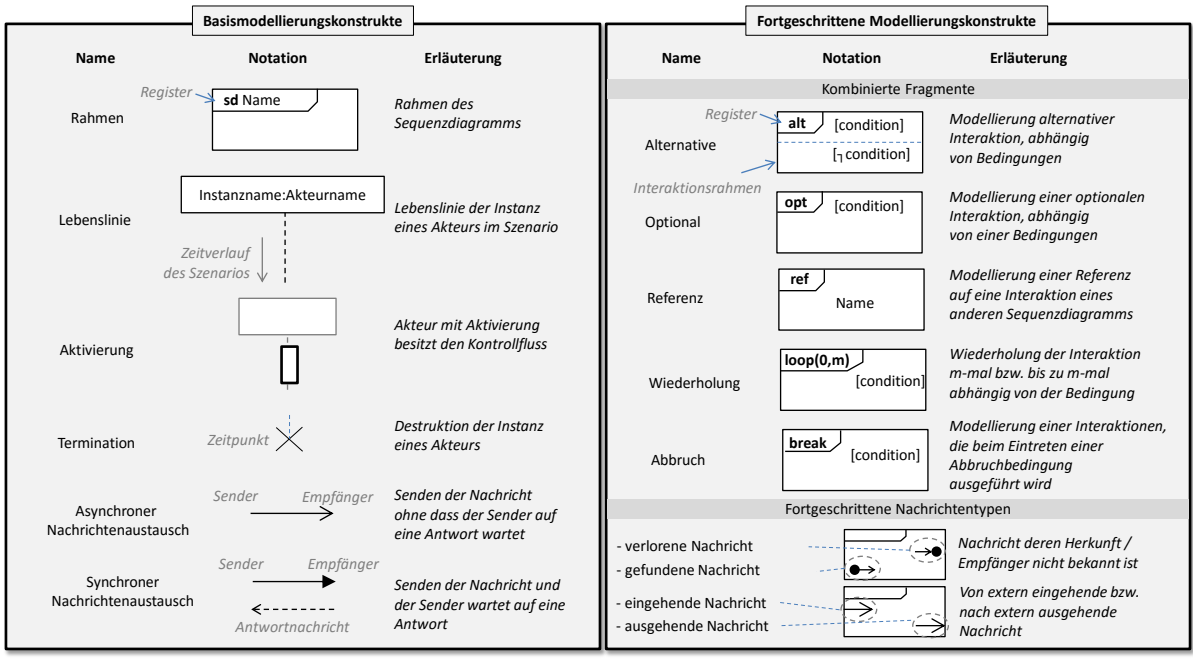

Abbildung 79: Modellierungskonstrukte zur Szenariomodellierung mit Sequenzdiagrammen

Im linken Bereich der Abbildung finden sich die Basismodellierungskonstrukte, d.h. solche Modellierungskonstrukte, die im Wesentlichen bei der Modellierung von Szenarien mit Sequenzdiagrammen zum Einsatz kommen. Im rechten Bereich der Abbildung sind solche Modellierungskonstrukte aufgeführt, die zur Modellierung umfangreicher und komplexer Interaktionsbeziehungen zwischen Kommunikationspartnern verwendet werden.

5.5.1 Basismodellierungskonstrukte

5.5.1.1 Modellierung der Identifizierbarkeit und Referenzierbarkeit des Szenarios

Sequenzdiagramme besitzen einen äußeren **Rahmen** (Interaktionsrahmen), der im oberen linken Bereich in einem Register den Namen des Szenarios beinhaltet, welches durch das jeweilige Sequenzdiagramm modelliert wird. Dem **Namen des Szenarios** ist in der Regel das Schlüsselwort „sd“ vorangestellt, welches, wie oben bereits erläutert, anzeigt, dass das Szenario in diesem Falle durch ein Sequenzdiagramm modelliert ist. Durch die Verwendung von Rahmen ist das Szenario über den Namen identifizierbar und referenzierbar, was insbesondere das Management der verschiedenen Diagramme unterstützt.

5.5.1.2 Modellierung der Kommunikationspartner des Szenarios

Eine **Lebenslinie** repräsentiert jeweils eine Instanz eines Akteurs innerhalb des Szenarios. Die Benennung der Lebenslinie erfolgt nach dem Muster `Instanzname:Typname` (z.B. `Peter:Fahrer`). Bei der Modellierung von Szenarien wird oft auf die Angabe von Instanznamen verzichtet. Instanznamen sollten allerdings dann angegeben werden, wenn dies die Verständlichkeit des modellierten Szenarios verbessert. Treten in einem Szenario beispielweise mehrere Instanzen eines bestimmen Kommunikationspartners auf, so sollte für jede Instanz ein konkreter Instanzname angegeben sein, weil durch diese Differenzierung u.a. deutlicher wird, dass zwei unterschiedliche Instanzen eines Akteurs am Szenario beteiligt sind, zwischen denen ggf. ein unmittelbarer Nachrichtenaustausch stattfindet. Die **Aktivierung** einer Lebenslinie zeigt an, dass der jeweilige Kommunikationspartner in dem entsprechend visualisierten Zeitabschnitt innerhalb des Szenarios die Kontrolle besitzt, d.h. den Kontrollfluss des Szenarios bestimmt. Wird in der Lebenslinie einer Instanz eine **Termination** modelliert, so charakterisiert dies die Destruktion der entsprechenden Instanz des Akteurs. Abbildung 80 zeigt ein Beispiel für die Modellierung einer Lebenslinie mit Aktivierung und Termination.

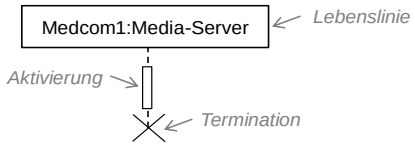

Abbildung 80: Modellierung von Lebenslinien und Termination

5.5.1.3 Beziehung von Akteuren in Szenarien zu Kontextmodellen und Use-Case-Modellen

Die in den Szenarien betrachteten Akteure treten wiederum in den Use-Case-Diagrammen des Systems und den Kontextdiagrammen auf, d.h. über die Kommunikationspartner innerhalb der Szenarien können die modellierten Szenarien mit den Use-Case-Diagrammen der Use-Case-Sicht (vgl. Abschnitt 4.2) und den Kontextdiagrammen integriert werden vgl. Abschnitt 2.2). Typischerweise entstehen die Kontextdiagramme im Vorfeld der Szenariomodellierung, so dass die im Kontextdiagramm dokumentierten Akteure und Schnittstellen die systematische Gewinnung und Dokumentation der Szenarien strukturieren und leiten können. Treten bei der Szenariomodellierung Akteure auf, die in den entsprechenden Use-Case-Diagrammen bzw. Kontextdiagrammen nicht vorzufinden sind, deutet dies auf eine Unvollständigkeit des Kontextmodells bzw. des Use-Modells hin (vgl. hierzu Abschnitt 4.2.3).

5.5.1.4 Modellierung des Nachrichtenaustausches innerhalb eines Szenarios

Der Nachrichtenaustausch zwischen zwei Instanzen von Kommunikationspartnern innerhalb eines Szenarios wird durch einen Pfeil visualisiert, wobei die Richtung des Pfeiles die Richtung des Nachrichtenaustausches anzeigt. Bezüglich des Nachrichtenaustausches wird zwischen zwei Arten unterschieden. Bei einem **asynchronen Nachrichtenaustausch** zwischen Instanzen innerhalb des Szenarios sendet der Sender die Nachricht an den Empfänger und wartet nicht auf eine zugehörige Reaktion in Form einer Nachricht des Empfängers. Asynchrone Nachrichten werden in der Szenariomodellierung u.a. dann verwendet, wenn eine Instanz eine oder mehrere Instanzen innerhalb des Szenarios informieren möchte und keine Rückantwort der Empfänger erwartet. Bei einem **synchronen Nachrichtenaustausch** zwischen Instanzen innerhalb eines Szenarios wartet der Sender der synchronen Nachricht auf eine Antwortnachricht des Empfängers. Synchrone Nachrichten werden in der Szenariomodellierung u.a. dann verwendet, wenn eine Instanz innerhalb des Szenarios eine Information von einer anderen Instanz anfordert. Ein Beispiel hierfür wäre die synchrone Nachricht „Erbitte Geheimzahl“ von der Instanz eines Geldautomaten zu der Instanz eines Benutzers, woraufhin der entsprechende Geldautomat darauf wartet, dass der Benutzer die Geheimzahl eingibt, d.h. eine Antwortnachricht mit der entsprechenden Geheimzahl sendet. Bei der Modellierung von Szenarien im Requirements Engineering bezieht sich der „Nachrichtenaustausch“ nicht nur auf Daten, die über eine Kommunikationsinfrastruktur zwischen Kommunikationspartnern übertragen werden. Ein „Nachrichtenaustausch“ innerhalb eines Szenarios kann auch den Austausch z.B. immaterieller oder materieller Dinge, wie z.B. das Einführen einer Kreditkarte (materieller Gegenstand) in den Geldautomaten durch den Benutzer, repräsentieren. Abbildung 81 zeigt jeweils ein Beispiel für die Modellierung asynchroner und synchroner Nachrichten.

Abbildung 81: Modellierung a) asynchroner und b) synchroner Nachrichten

Durch einen Nachrichtenaustausch kann der sendende Kommunikationspartner von einem anderen Kommunikationspartner einen Dienst anfordern. Der **Dienstaufruf** kann dabei wiederum asynchron oder synchron geschehen. Beim asynchronen Aufruf eines Dienstes wird der Dienst mittels einer Nachricht lediglich angestoßen, d.h. der aufrufende Kommunikationspartner wartet nicht auf eine Antwort. Bei einem synchronen Aufruf wartet der Sender, nachdem er mittels einer Nachricht den Dienst von einem anderen Kommunikationspartner angefordert hat, auf die entsprechende Rückantwort des Empfängers. Bei einem Dienstaufruf kann zusätzlich dessen Signatur, d.h. **Eingabeparameter** (Argumente) und **Rückgabeparameter** angegeben werden. Parameter sind dabei typischerweise innerhalb der Informationsstruktursicht definiert, wodurch eine Verbindung (Integration) zwischen der Szenariosicht und der Informationsstruktursicht hergestellt werden kann. Die Abbildung zeigt darüber hinaus die Verwendung des optionalen Modellierungskonstrukts für die Repräsentation der Aktivierung eines Kommunikationspartners. Abbildung 82 zeigt jeweils ein Beispiel für die Modellierung eines Dienstaufrufes mit unvollständigen und vollständigen Parameter.

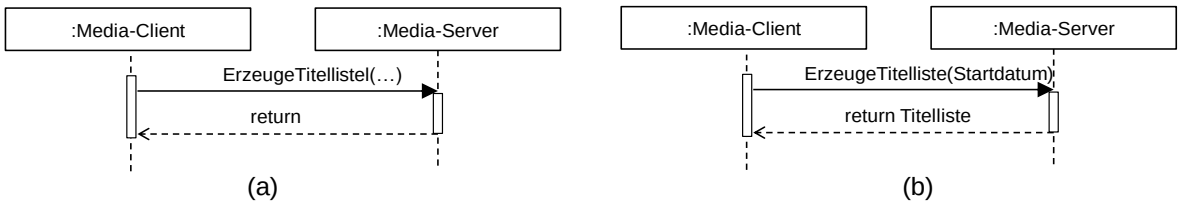

Abbildung 82: Modellierung eines Dienstaufrufes a) mit unvollständigen und b) vollständigen Parameter

5.5.1.5 Beziehung von Nachrichten in Szenarien zur zustandsbasierten, datenflussorientierten Modellierung und Informationsstrukturmodellierung

Der Austausch von Nachrichten innerhalb eines Szenarios stellt den wesentlichen Integrationspunkt zu den Diagrammen anderer Sichten auf die Anforderungen des betrachteten Systems dar (vgl. Abbildung 83).

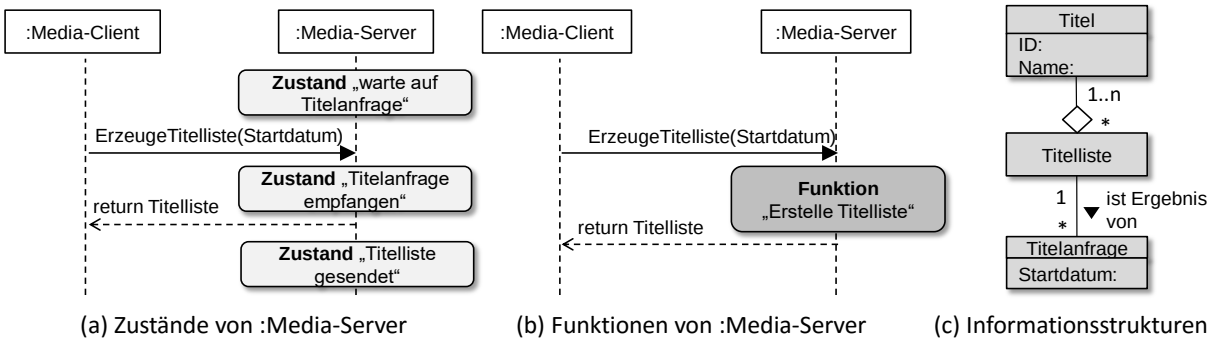

Abbildung 83: Nachrichten in Szenarien als Integrationspunkt mit anderen Anforderungssichten

5.5.1.5.1 Bezug von Nachrichten zu Zuständen in der zustandsorientierten Sicht

Wie in Abbildung 83 (a) gezeigt, geht sowohl das Empfangen als auch das Senden einer Nachricht mit einem Zustandswechsel beim zugehörigen Akteur einher, d.h. in Abbildung 83 (a) geht z.B. das Empfangen der Nachricht „NeueTitel(Startdatum)“ mit einem Zustandswechsel des Kommunikationspartners „:Media-Server“ einher, weil dieser Kommunikationspartner z.B. vom Zustand „Warte auf Titelanfrage“ in den Zustand „Titelanfrage empfangen“ wechselt. Auch das Versenden der Nachricht „return Titelliste“ führt wiederum zu einem Zustandswechsel von „:Media-Sever“(in den Zustand „Titelliste gesendet“). Gleichzeitig führt auch das Empfangen dieser Nachricht beim Empfänger „:Media-Client“ zu einem Zustandswechsel. Beispielsweise können die Zustände der verschiedenen Kommunikationspartner eines Szenarios und die Zustandsübergänge durch Diagramme der zustandsorientierten Sicht modelliert werden, z.B. durch ein Zustandsmaschinendiagramm der UML (vgl. auch Abschnitt 4.4).

5.5.1.5.2 Bezug von Nachrichten zu Funktionen/Aktivitäten in der datenflussorientierten bzw. kontrollflussorientierten Sicht

Wie in Abbildung 83 (b) gezeigt, besteht bezogen auf das zu entwickelnde System zwischen dem Empfang einer Nachricht und dem anschließenden Senden einer Nachricht ein funktionaler Zusammenhang, da das System eine Funktion ausführen muss, um auf Basis der eingehenden Nachricht und ggf. lokal verfügbaren Informationen die Ergebnisnachricht zu erzeugen. Diese Funktionen (Prozesse, Aktivitäten) werden typischerweise in der datenflussorientierten bzw. kontrollflussorientierten Sicht modelliert, indem die Datenabhängigkeiten bzw. Kontrollflussabhängigkeiten zwischen solchen Systemfunktionen z.B. in einem oder

mehreren Datenflussdiagrammen bzw. Aktivitätsdiagrammen modelliert werden (vgl. auch Abschnitt 4.3).

5.5.1.5.3 Bezug von Nachrichten zu Klassen, Attributen und Assoziationen in der Informationsstruktursicht

Wie in Abbildung 83 (c) gezeigt, werden die Nachrichten und ggf. zugehörigen Paramter in der Informationsstruktursicht der Anforderungen definiert. Da heißt, dass entsprechende Informationen z.B. in einem Klassendiagramm spezifiziert sind, welches im Detail die Informationsstruktur der Nachrichten definiert sowie den fachlichen Bezug zu anderen Nachrichten herstellt, die zwischen dem zu entwickelnden System und den Akteuren im Systemkontext ausgetauscht werden (vgl. auch Abschnitt 3).

5.5.2 Fortgeschrittene Modellierungskonstrukte

Die Verwendung kombinierter Fragmente ermöglicht es, umfangreiches und komplexes interaktionsbasiertes Verhalten in Szenarien auf leicht verständliche Weise durch Sequenzdiagramme zu modellieren. Die UML bzw. SysML unterscheiden eine Vielzahl unterschiedlicher Arten kombinierter Fragmente. Im Folgenden werden fünf Arten kombinierter Fragmente betrachtet, die sich sehr gut zur Modellierung von umfangreichem und komplexem interaktionsbasiertem Verhalten in Szenarien eignen. Kombinierte Fragmente werden durch Interaktionsrahmen innerhalb des Sequenzdiagramms modelliert. Der Typ des jeweiligen kombinierten Fragments und damit die zugehörige Bedeutung, in der die innerhalb des kombinierten Fragments modellierte Interaktion zum umgebenden Szenario steht, wird jeweils durch ein Schlüsselwort innerhalb des kombinierten Fragments angegeben. In der vertikalen Dimension des Sequenzdiagramms (Zeitverlauf) wird der Interaktionsrahmen typischerweise so weit ausgeprägt, wie die spezifischen Interaktionen im Zeitverlauf auftreten. In der horizontalen Dimension sind die Interaktionsrahmen der kombinierten Fragmente soweit ausgeprägt, dass sie sämtliche Instanzen einschließen, zwischen denen hinsichtlich der spezifischen Interaktion innerhalb des kombinierten Fragments ein Nachrichtenaustausch auftritt.

5.5.2.1 Modellierung alternativer Interaktionen eines Szenarios (alt)

Zur Modellierung alternativer Interaktionsfolgen (d.h. eines alternativen Verhaltens) von Szenarien werden Alternative Fragmente verwendet. Hierzu wird innerhalb des Sequenzdiagramms ein entsprechender Interaktionsrahmen modelliert, der im Register durch das Schlüsselwort „alt“ gekennzeichnet ist. Der Interaktionsrahmen ist in zwei oder mehr Abschnitte unterteilt, wobei, um die Anforderungen möglichst präzise spezifizieren zu können, für jeden dieser Abschnitte eine explizite boolesche Bedingung angegeben sein muss, die festlegt, wann („wann“ im Sinne einer logischen Bedingung) die im zugehörigen Abschnitt enthaltene Interaktion ausgeführt wird. Für einen Abschnitt kann auch die Bedingung „else“ angegeben werden, wodurch festlegt wird, dass die zugehörige Interaktion dann ausgeführt wird, wenn keine der anderen Bedingungen zum Zeitpunkt des möglichen Eintritts in das kombinierte Fragment wahr ist. Wird auf die explizite Modellierung eines solchen Abschnitts verzichtet, so wird dann, wenn beim Eintritt in das kombinierte Fragment keine der Bedingungen wahr ist, auch keine Interaktion ausgeführt. Die boolesche Bedingung eines jeden Abschnitts wird typischerweise über der Lebenslinie derjenigen Instanz innerhalb des Szenarios modelliert, die Zugriff auf den Wert hat, mit dem die boolesche Bedingung ausgewertet werden kann. Handelt es sich um globale Werte, so kann die boolesche Bedingung beliebig über den Lebenslinien angeordnet werden. Bei der Formulierung der Bedingungen für einzelne Abschnitte der alternativen Interaktion des Szenarios ist darauf zu achten, dass die-

se aus logischer Sicht überschneidungsfrei sind, d.h. dass beim Eintritt in das entsprechende kombinierte Fragment nicht mehr als eine Bedingung wahr ist, da ansonsten das zugehörige Szenario ein nicht-deterministisches Verhalten aufweist (siehe Abschnitt 4.4). Abbildung 84 zeigt ein Beispiel für die Modellierung eines kombinierten Fragments vom Typ „Alternative“.

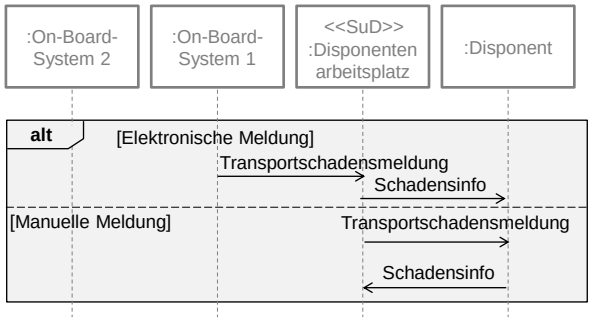

Abbildung 84: Modellierung eines kombinierten Fragments des Typs „Alternative“

5.5.2.2 Modellierung optionaler Interaktionen eines Szenarios (opt)

Die Modellierung optionaler Interaktionen (d.h. eines optionalen Verhaltens) von Szenarien werden Optionale Fragmente verwendet. Hierzu wird innerhalb des Sequenzdiagramms ein entsprechender Interaktionsrahmen modelliert, der im Register durch das Schlüsselwort „opt“ gekennzeichnet ist. Im Interaktionsrahmen sollte eine explizite boolesche Bedingung angegeben sein, die festlegt, welche Bedingung beim Ablauf des Szenarios zum Zeitpunkt des möglichen Eintritts in das kombinierte Fragment wahr sein muss, damit die im optionalen Fragment modellierte Interaktion während des Ablaufs des Szenarios ausgeführt werden. Die boolesche Bedingung wird typischerweise über der Lebenslinie derjenigen Instanz innerhalb des Szenarios modelliert, die bestimmt, ob die entsprechende Bedingung erfüllt ist oder nicht. Ist die Bedingung beim möglichen Eintritt in das kombinierte Fragment nicht wahr, so tritt die entsprechende Interaktion (bzw. der zugehörige Nachrichtenaustausch) während der Ausführung des Szenarios nicht auf. Ein optionales kombiniertes Fragment kann als ein alternatives kombiniertes Fragment angesehen werden, das nur einen Abschnitt und eine zugehörige Bedingung besitzt. Abbildung 85 zeigt ein Beispiel für die Modellierung eines kombinierten Fragments vom Typ „Optional“.

Abbildung 85: Modellierung eines kombinierten Fragments des Typs „Optional“

5.5.2.3 Modellierung von Abstraktionen über Interaktionsfolgen eines Szenarios (ref)

Sequenzdiagramme bieten die Möglichkeit, von zusammenhängenden Interaktionsfolgen eines Szenarios zu abstrahieren, indem an der entsprechenden Stelle im Sequenzdiagramm auf ein anderes Sequenzdiagramm verwiesen wird, welches die entsprechende Interaktion des Szenarios modelliert. Hierzu wird innerhalb des Sequenzdiagramms an derjenigen Stelle, an der die abstrahierte Interaktion auftritt, ein kombiniertes Fragment modelliert, welches im Register mit dem Schlüsselwort „ref“ gekennzeichnet ist. Im Zentrum des Fragments wird der Name des Szenarios angegeben, in welchem die detaillierte Interaktion enthalten

ist, die bei der Ausführung des übergeordneten Szenarios an der durch das kombinierte Fragment angezeigten Position in die Interaktion des Szenarios integriert wird. Die Verwendung von kombinierten Fragmenten dieses Typs bietet sich insbesondere dann an, wenn umfangreiches oder komplexes Interaktionsverhalten eines Szenarios modelliert werden muss. Hierdurch kann der Requirements Engineer fachlich zusammenhängende Interaktionen eines komplexen Szenarios in ein separates Sequenzdiagramm auslagern. Die Verwendung kombinierter Fragmente des Typs „Referenz“ bietet sich auch dann an, wenn gewisse Interaktionen (wie z.B. die Interaktionen für die Authentifizierung eines Benutzers am System) auf identische Weise in mehreren Szenarien auftreten. Bei der Modellierung von Interaktionsfolgen in separaten Sequenzdiagrammen, auf die in anderen Sequenzdiagrammen mittels eines kombinierten Fragments des Typs „Referenz“ verwiesen wird, muss der Requirements Engineer darauf achten, dass das zu inkludierende Teilszenario mit dem umgebenden Szenario verträglich ist. So dürfen in den Teilszenarien beispielsweise keine Instanzen auftreten, die in dem umgebenden Szenario bzw. dem zugehörigen Sequenzdiagramm nicht vorhanden sind. Abbildung 86 zeigt ein Beispiel für die Modellierung eines kombinierten Fragments vom Typ „Referenz“.

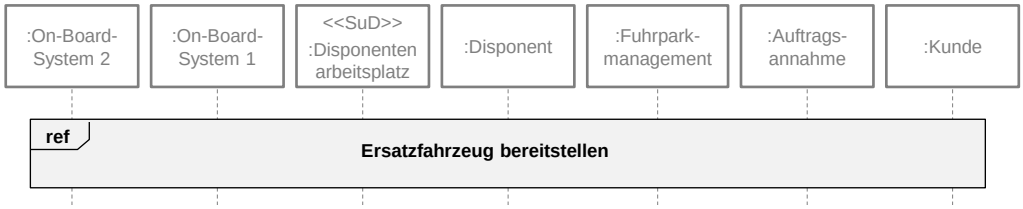

Abbildung 86: Modellierung eines kombinierten Fragments des Typs „Referenz“

5.5.2.4 Modellierung der Wiederholung von Interaktionen innerhalb eines Szenarios (loop)

Um Wiederholungen von Interaktionen innerhalb eines Szenarios ausdrücken zu können, wird innerhalb des Sequenzdiagramms des Szenarios ein Interaktionsrahmen modelliert, der im Register durch das Schlüsselwort „loop“ gekennzeichnet ist. In kombinierten Fragmenten dieses Typs wird die Anzahl der Wiederholungen entweder fest durch `loop ([Anzahl])` angegeben oder mit `loop([Min,Max])` eine untere (min) und obere (max) Schranke für die Anzahl der Wiederholungen festgelegt. Im letzten Falle drückt die Bedingung für die Wiederholung aus, dass die Interaktion innerhalb des Interaktionsrahmens mindestens min-Mal und höchstens max-Mal wiederholt wird. Die Wiederholung der Interaktion innerhalb des Interaktionsrahmens wird dabei zusätzlich noch durch eine boolesche Bedingung bestimmt. Wenn die zu wiederholende Interaktion des Szenarios min-Mal ausgeführt wurde, wird die Wiederholung dann abgebrochen, wenn die Auswertung der booleschen Bedingung beim erneuten Eintritt in den Interaktionsrahmen des kombinierten Fragments falsch ist. Ist die boolesche Bedingung bei jedem der Eintritte in den Interaktionsrahmen wahr, so wird die Wiederholung der Interaktion nach max-Durchläufen beendet. Abbildung 87 zeigt ein Beispiel für die Modellierung eines kombinierten Fragments vom Typ „Loop“.

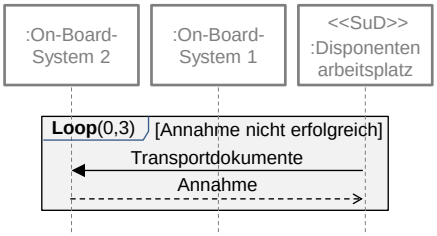

Abbildung 87: Modellierung eines kombinierten Fragments des Typs „Loop“

5.5.2.5 Modellierung des Abbruchs eines Szenarios (break)

Während des Ablaufes eines Szenarios können Situationen eintreten, die eine erfolgreiche Durchführung des Szenarios verhindern. Um in solchen Fällen die aus fachlicher Sicht notwendige Ausnahmebehandlung zu betrachten, kann in Sequenzdiagrammen die Interaktion zur Ausnahmebehandlung ebenfalls modelliert werden. Das Abbruchfragment besitzt optional eine boolesche Bedingung und optional eine Interaktion, die beim Eintritt der Ausnahmebedingung zur definierten Behandlung des Abbruchs ausgeführt wird. Ist keine explizite Abbruchbedingung formuliert, so dokumentiert das Abbruchfragment lediglich die Interaktion, die ausgeführt wird, wenn eine nicht näher spezifizierte Abbruchbedingung eintritt. Zur präzisen Spezifikation von Anforderungen ist es allerdings zwingend notwendig, dass die Abbruchbedingungen explizit dokumentiert sind. Kommt es während des Ablaufs des entsprechenden Szenarios zu einem Abbruch, so wird lediglich noch die Interaktion im Abbruchfragment ausgeführt, d.h. der Ablauf des Szenarios endet nach Ausführung der Interaktion im Abbruchfragment, auch wenn innerhalb des Sequenzdiagramms nach dem Abbruchfragment noch weitere Interaktionen folgen. Diese Interaktionen werden dann ausgeführt, wenn im Ablauf des Szenarios die Abbruchbedingung nicht eintritt. Besitzt ein Abbruchfragment keine Interaktion, so terminiert das Szenario unmittelbar den Eintritt der Abbruchbedingung. Abbildung 88 zeigt ein Beispiel für die Modellierung eines kombinierten Fragments vom Typ „Break“.

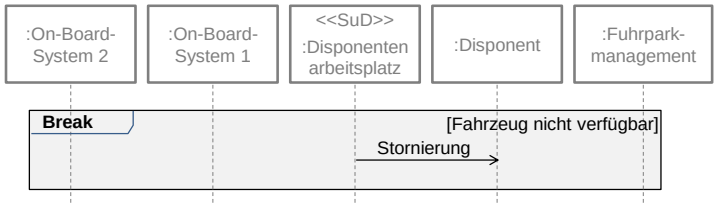

Abbildung 88: Modellierung eines kombinierten Fragments des Typs „Break“

5.5.3 Schachtelung von Fragmenten

Durch die Verwendung kombinierter Fragmente ist es möglich, in einem einzelnen Sequenzdiagramm mehrere potenzielle Abläufe eines Szenarios zu modellieren. Dies ist insbesondere dann der Fall, wenn kombinierte Fragmente geschachtelt werden. Beispielsweise ergeben sich durch die Verwendung eines einzelnen Alternativen Fragments mit drei alternativen Interaktionsfolgen mindestens drei mögliche Abläufe des Szenarios. Im Falle der Verwendung eines einzelnen Optionalen Fragments ergeben sich mindestens zwei mögliche Abläufe des Szenarios, ein bestimmter Ablauf wenn die entsprechende Bedingung zur Ausführung der Interaktion im Optionalen Fragment gilt und ein anderer Ablauf wenn diese Bedingung nicht gilt. Tritt innerhalb einer Alternative in einem kombinierten Fragment des Typ „Alternative“ wiederum ein kombiniertes Fragment des Typs „Optional“ auf, so ergeben in Bezug auf den Nachrichtenaustausch innerhalb der alternativen Interaktion wiederum zwei mögliche Ab-

läufe des Szenarios. In ähnlicher Weise gilt dies auch für die Verschachtelung anderen Typen von Fragmenten. Sequenzdiagramme, die solche kombinierten Fragmente enthalten, dokumentieren demnach mehrere mögliche Abläufe hinsichtlich des betrachteten Szenarios. Dadurch lassen sich mittels Sequenzdiagrammen auf verständliche Weise zusammenhängende Haupt-, Alternativ- und Ausnahmeszenarien (Abbruchszenarien) modellieren. Haupt-, Alternativ- und Ausnahmenszenarien sind in diesem Fall jeweils durch einen Kontrollflusspfad des Szenarios definiert. Abbildung 89 zeigt ein Beispiel für die Modellierung verschachtelter kombinierter Fragmente.

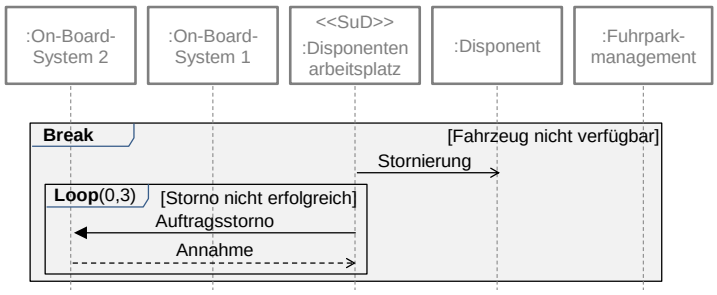

Abbildung 89: Modellierung verschachtelter kombinierter Fragmente

5.5.4 Modellierung von Annahmen eines Szenarios

Szenarien basieren in der Regel auf einer Vielzahl von Annahmen, deren Gültigkeit vorausgesetzt wird, damit das Szenario tatsächlich in der modellierten Art und Weise ausgeführt werden kann. Werden Szenarien in Sequenzdiagrammen modelliert, so können bspw. die Annahmen in Form textueller Annotationen definiert und durch geeignete Abhängigkeitsbeziehungen zu den Modellelementen innerhalb des Szenarios relationiert werden, auf die sich die Annahmen beziehen. Abbildung 90 zeigt an einem einfachen Beispiel die Modellierung von Annahmen, die einem Szenario zugrunde liegen.

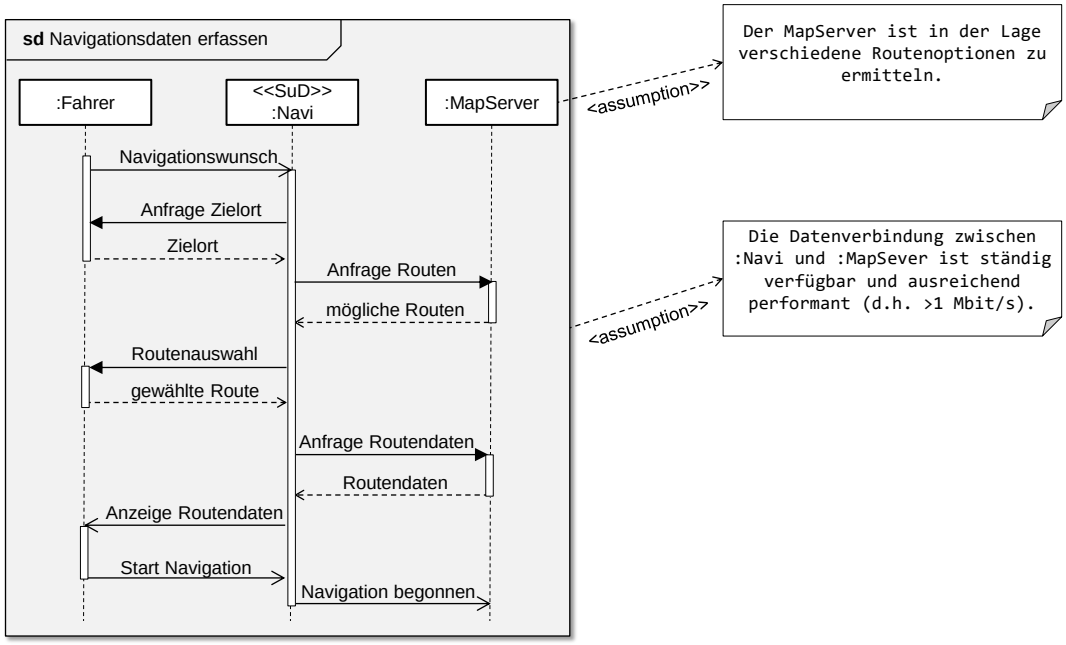

Abbildung 90: Modellierung von Annahmen zu einem Szenario

Der Bezug zwischen Modellelementen des Sequenzdiagramms und den zugehörigen Annahmen wird hier durch eine gerichtete Abhängigkeitsbeziehung mit dem Stereotyp <<assumption>> hergestellt (vgl. Abschnitt 1.8). Wie in der Abbildung gezeigt, können die An-

nahmen zum gesamten Szenario oder auch zu einzelnen Modellelementen innerhalb des Szenarios, auf die sich diese Annahme bezieht, relationiert werden. Die Aussage einer solchen Annahme ist, dass z.B. im Falle der Annahme zum Akteur :MapServer das Szenario nur dann erfolgreich ausgeführt werden kann, wenn der :MapServer die Annahme erfüllt. Somit können in einem Szenario bspw. Fehlerfälle ausgeschlossen werden, die nicht zum allgemeinen Verständnis des Ablaufes beitragen.

5.6 Szenariomodellierung mit Kommunikationsdiagrammen

Abbildung 91 zeigt die Modellierungskonstrukte der UML für Kommunikationsdiagramme, die zur Modellierung von Szenarien verwendet werden. Kommunikationsdiagramme besitzen ebenfalls einen äußeren **Rahmen**, der im oberen linken Bereich in einem Register den Namen des Szenarios beinhaltet, welches durch das jeweilige Kommunikationsdiagramm modelliert wird. Dem **Namen des Szenarios** ist in der Regel das Schlüsselwort „cm“ vorangestellt, welches anzeigt, dass das Szenario in diesem Falle durch ein Kommunikationsdiagramm modelliert ist. Eine **Lebenslinie** repräsentiert jeweils eine Instanz eines Akteurs innerhalb des Szenarios. Die Benennung der Lebenslinie erfolgt nach dem Muster `Instanzname:Typname` (z.B. `Peter:Fahrer`). Findet innerhalb des betrachteten Szenarios ein unmittelbarer **Nachrichtenaustausch** zwischen zwei Instanzen statt, so wird dies im Kommunikationsdiagramm durch eine Verbindungslinie zwischen diesen Instanzen modelliert.

Name	Notation	Erläuterung
Rahmen	**cm** Name	*Rahmen des Kommunikationsdiagramms*
Lebenslinie	:Name	*Lebenslinie eines Akteurs im Szenario*
Nachrichtenaustausch	———	*Modelliert einen generellen Nachrichtenaustausch Zwischen Akteuren*
Kommunikationsrichtung	⟶	*Modelliert die Richtung eines Nachrichtenaustauschs*
Nachrichtensignatur	**Sequenznummer:Nachricht**	*Eine Nachricht im Szenario wird abhängig von der Reihenfolge des Auftretens im Szenario mit einer Sequenznummer versehen*

Abbildung 91: Modellierungskonstrukte von Kommunikationsdiagrammen zur Szenariomodellierung

Jede Nachricht, die zwischen Instanzen innerhalb des Szenarios ausgetauscht wird, wird an der entsprechenden Verbindungslinie in Form einer **Nachrichtensignatur** annotiert. Die Nachrichtensignatur umfasst die eigentliche **Nachricht** sowie die **Sequenznummer** des Nachrichtenaustauschs innerhalb des Szenarios. Die **Kommunikationsrichtung** einer Nachricht wird jeweils durch einen Pfeil angezeigt.

5.7 Beispiele für typische Diagramme in der Szenariosicht

Mit Hilfe der verschiedenen Arten kombinierter Fragmente können komplexe Interaktionen zwischen Akteuren sowie zwischen Akteuren und dem betrachteten System modelliert werden. Tabelle 4 fasst typische Einsatzbereiche kombinierter Fragmente in der Szenariomodellierung sowie bei der Betrachtung von Szenarien innerhalb von Use-Cases zusammen.

Szenario-Ebene	Szenarien auf der Use-Case-Ebene	Fragment
Modellierung alternativer Folgen von Nachrichten zwischen Kommunikationspartnern	Modellierung alternativer Extend-Beziehungen zwischen Use-Cases an einem Extension-Point	Alt
Modellierung optionaler Nachrichten zwischen Kommunikationspartnern	Modellierung einzelner Extend-Beziehung zwischen Use-Cases, die keine Ausnahmebehandlung betrachten	Opt
Abstraktion über eine zusammenhängende Folge von Nachrichten z.B. zur Komplexitätsbeherrschung und Verbesserung der Lesbarkeit	Modellierung von Include-Beziehungen zwischen Use-Cases	Ref
Modellierung bedingungsabhängiger Wiederholung von Nachrichten zwischen Kommunikationspartnern innerhalb von Szenarien	—	Loop
Modellierung von Ausnahmebehandlung in Szenarien	Ausnahmebehandlung durch Extend-Beziehungen zwischen Use-Cases	Break

Tabelle 4: Typische Einsatzzwecke kombinierter Fragmente in der Szenariomodellierung

In diesem Abschnitt wird der Einsatz der obigen Typen kombinierter Fragmente im Rahmen der Szenariomodellierung anhand exemplarischer Ausschnitte aus der Szenariosicht auf die Software eines Disponentenarbeitsplatzes im Transportmanagement verdeutlicht.

5.7.1 Modellierung von Szenarien durch Sequenzdiagramme

Abbildung 92 und Abbildung 93 zeigen Ausschnitte aus der Szenariosicht für einen Disponentenarbeitsplatz in Form zweier Sequenzdiagramme der UML/SysML. In dem in Abbildung 92 dargestellten Sequenzdiagramm ist das Szenario „Ersatzfahrzeug bereitstellen“ modelliert, welches die Interaktion zwischen den Instanzen `:On-Board-System 2`, `On-Board-System 1`, `Disponentenarbeitsplatz`, `Disponenten`, `Fuhrparkmanagement` und `Auftragsannahme` modelliert, die ablaufen müssen, damit ein Ersatzfahrzeug bereitgestellt werden kann. Der Disponentenarbeitsplatz stellt hier das zu entwickelnde Softwaresystem dar, die anderen Kommunikationspartner im Szenario sind Instanzen von Akteuren im Systemkontext des Disponentenarbeitsplatzes.

Das gezeigte Szenario greift neben Basismodellierungskonstrukten zur Szenariomodellierung mit Sequenzdiagrammen der UML/SysML auch auf fortgeschrittene Modellierungskonstrukte in Form von zwei Wiederholungsfragmenten (Schlüsselwort „loop“) und eines Abbruchfragments (Schlüsselwort „break“) zurück. Das in der Interaktion als erstes auftretende Wiederholungsfragment modelliert dabei, dass maximal dreimal der Versuch der Übersendung der Transportdokumente durch den Disponentenarbeitsplatz unternommen wird. Nach dem Senden der Transportdokumente durch den Disponentenarbeitsplatz wartet dieser auf die Annahme der Transportdokumente durch das On-Board-System des Ersatzfahrzeuges (d.h. eine synchrone Nachricht). Diese Interaktion wird ausgeführt, solange die Bedingung „Annahme nicht erfolgreich“ wahr ist. Gilt diese Bedingung beim Eintritt in das kombinierte Fragment nicht, so wird die entsprechende Interaktion im kombinierten Fragment nicht mehr ausgeführt. Im Ablauf des Szenarios wird im nächsten Schritt vom Disponentenarbeitsplatz die asynchrone Nachricht „Fahrzeugwahl“ zum Disponenten geschickt.

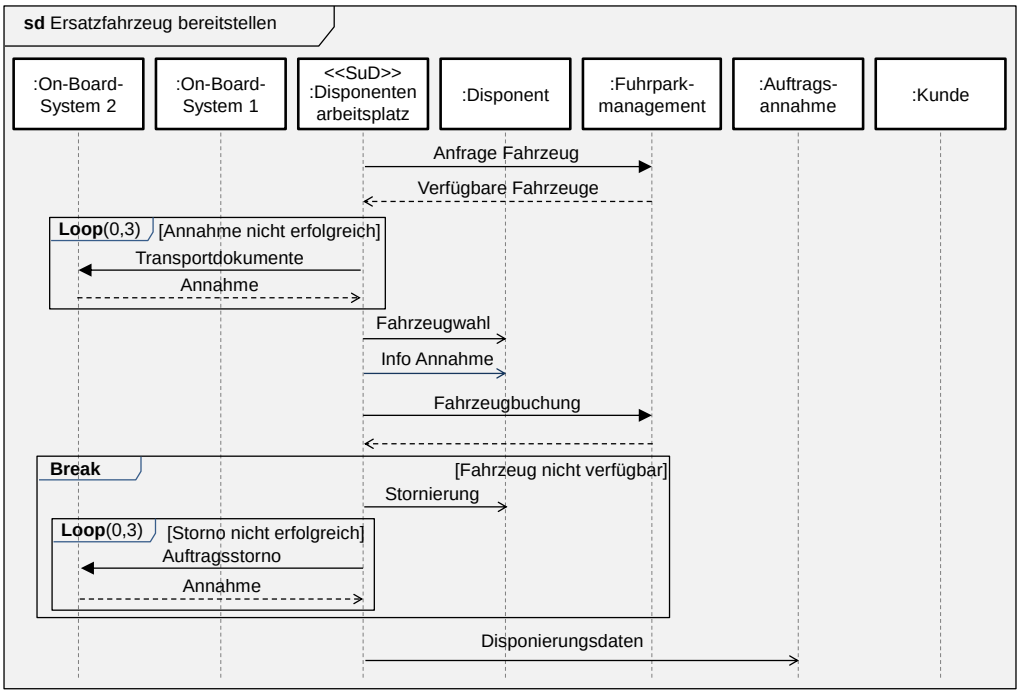

Abbildung 92: Beispiel für durch Sequenzdiagramme modellierte Szenarien

Das Abbruchfragment dokumentiert, dass beim Auftreten der Bedingung „Fahrzeug nicht verfügbar“ eine asynchrone Nachricht vom Disponentenarbeitsplatz zum Disponenten gesendet und eine Interaktion zur Auftragsstornierung zwischen dem Disponentenarbeitsplatz und dem On-Board-System des Ersatzfahrzeuges abläuft, die maximal dreimal wiederholt wird. Ist beim Eintritt in dieses Fragment die Bedingung „Storno nicht erfolgreich“ ungültig, d.h. das Storno war erfolgreich, wird die Interaktion innerhalb des Wiederholungsfragments nicht wieder ausgeführt. Wurde das Abbruchfragment betreten, terminiert das Szenario nach der Ausführung der Interaktion innerhalb des Abbruchfragments, d.h. die asynchrone Nachricht „Disponierungsdaten“ vom Disponentenarbeitsplatz zur Auftragsannahme wird nicht mehr gesendet.

In dem in Abbildung 93 modellierten Sequenzdiagramm ist das Szenario „Ersatzlieferung bei Transportschaden“ modelliert, welches die Interaktion zwischen den Instanzen `:On-Board-System 2`, `On-Board-System 1`, `Disponentenarbeitsplatz`, `Disponenten`, `Fuhrparkmanagement`, `Auftragsannahme` und `Kunde` modelliert, die ablaufen muss, damit bei einem eingetretenen Transportschaden eine Ersatzlieferung avisiert werden kann. Zur Modellierung des Szenarios „Ersatzlieferung bei Transportschaden“ wurden ebenfalls verschiedene fortgeschrittene Modellierungskonstrukte der Szenariomodellierung mit Sequenzdiagrammen verwendet. Beispielsweise ist durch das Alternative Fragment zu Beginn modelliert, dass wenn die elektronische Meldung eines Transportschadens erfolgt, die Transportschadensmeldung vom On-Board-System des Schadenfahrzeuges an den Disponentenarbeitsplatz gesendet wird und dieser dann eine Nachricht mit den Schadeninformationen an den Disponenten sendet.

Alternativ dazu kann die Meldung eines Transportschadens über andere Wege den Disponenten erreichen. In diesem Falle kommt auf anderem Wege die Nachricht über einen eingetretenen Transportschaden (→ Found Message) direkt an den Disponenten, woraufhin der Disponent die notwendigen Schadensinformationen zur Weiterverarbeitung am Disponentenarbeitsplatz erfassen muss.

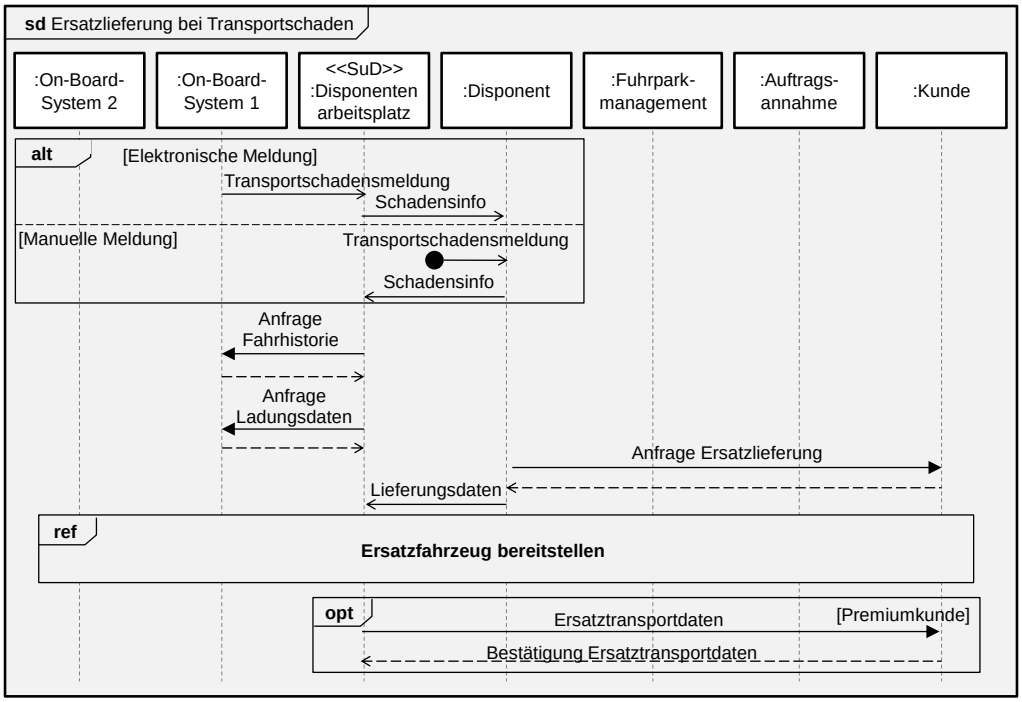

Abbildung 93: Beispiel für durch Sequenzdiagramme modellierte Szenarien

Das Referenzielle Fragment im unteren Bereich des Sequenzdiagramms dokumentiert, dass an dieser Stelle im Ablauf des Szenarios die Interaktion des Szenarios „Ersatzfahrzeug bereitstellen“ (Abbildung 92) inkludiert wird. Das Optionale Fragment im unteren Bereich beschreibt, dass innerhalb des Szenarios eine Nachricht mit den Ersatztransportdaten vom Disponentenarbeitsplatz zum Kunden gesendet wird und der Disponentenarbeitsplatz auf eine entsprechende Bestätigung wartet. Dies geschieht aber nur dann, wenn die Bedingung „Premiumkunde“ gilt, d.h. wenn es sich bei dem Transportkunden um einen Premiumkunden handelt. Ist dies nicht der Fall, terminiert das Szenario bereits mit Ende der inkludierten Interaktionen des Szenarios „Ersatzfahrzeug bereitstellen“.

5.7.2 Modellierung von Szenarien durch Kommunikationsdiagramme

Abbildung 94 zeigt einen Ausschnitt aus der Szenariosicht für den Disponentenarbeitsplatz in Form eines Kommunikationsdiagramms der UML, welches das Szenario „Ersatzfahrzeug bereitstellen“ des Disponentenarbeitsplatz modelliert (vgl. hierzu auch Abbildung 92). Wie aus der Abbildung ersichtlich wird, eignen sich Kommunikationsdiagramme kaum dazu, komplexes interaktionsbasiertes Verhalten von Szenarien zu modellieren, da dieser Diagrammtyp nicht über Modellierungskonstrukte verfügt, durch die z.B. optionale oder alternative Interaktionsfolgen von Szenarien modelliert werden können. Darüber hinaus verfügen Kommunikationsdiagramme auch nicht über Modellierungskonstrukte, die es gestatten, Teile von Interaktionsfolgen zu abstrahieren, indem die entsprechenden Interaktionen des Szenarios in einem gesonderten Diagramm modelliert werden, auf das im übergeordneten Diagramm verwiesen werden kann. Nichtdestotrotz sind Kommunikationsdiagramme dann von Vorteil, wenn der Fokus der Betrachtung auf dem bilateralen Nachrichtenaustausch zwischen Instanzen eines Szenarios liegt.

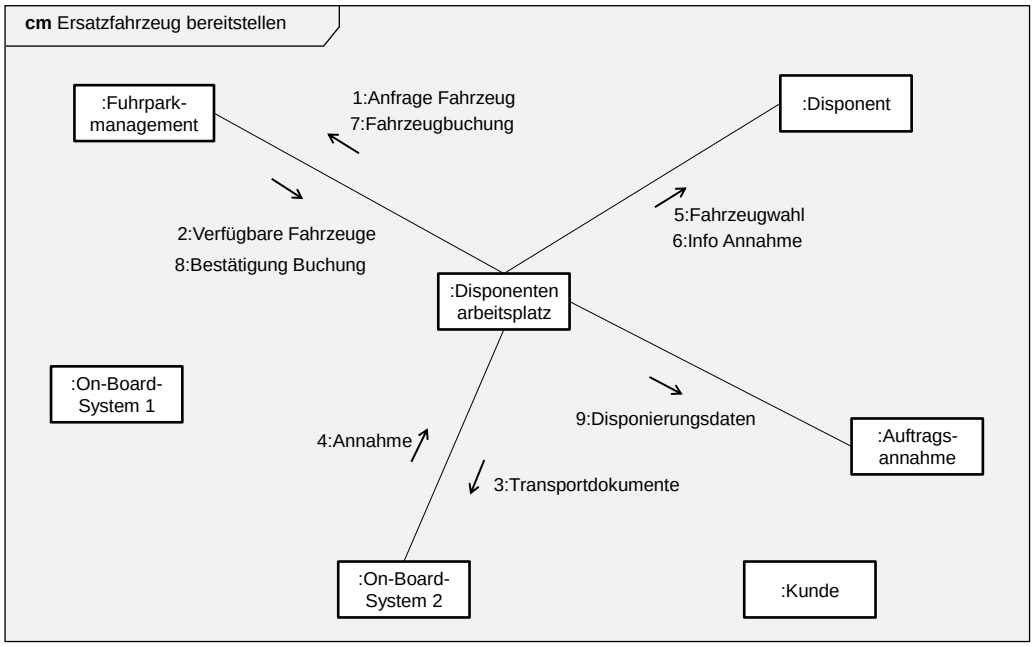

Abbildung 94: Beispiel für durch ein Kommunikationsdiagramm modelliertes Szenario

Wenn also der Requirements Engineer ein Szenario unter diesem Fokus modellieren möchte, weil z.B. die Eigenschaften der bilateralen Schnittstellen (Mensch-Maschine bzw. Maschine-Maschine) zwischen dem zu entwickelnden System und den Instanzen von Akteuren im Mittelpunkt stehen, bietet sich die Verwendung dieses Diagrammtyps ggf. ergänzend zur Modellierung des Szenarios mit Sequenzdiagrammen an.

5.8 Vertiefende Literatur

Typen von Szenarien und deren Dokumentation

- Rolland, C.; Achour, C.; Cauvet, C.; Ralyté, J.; Sutcliffe, A.; Maiden, N.; Jarke, M.; Haumer, P.; Pohl, K.; Dubois, E.; Heymans, P.: A Proposal for a Scenario Classification Framework. In: Requirements Engineering, 3 (1998) 1, S.23-47
- Jacobson, I.; Christerson, M.; Jonsson, P.; Oevergaard, G.: Object Oriented Software Engineering – A Use-Case Driven Approach. Addison-Wesley, Reading, 1992.

Szenariomodellierung im Requirements Engineering

- Pohl, K.: Requirements Engineering – Fundaments, Principles, Techniques. Springer, 2010.

Modellierung von Sequenz- und Kommunimkationsdiagrammen

- Object Management Group: OMG Systems Modeling Language (OMG SysML) Language Specification v1.2. OMG Document Number: formal/2010-06-02.
- Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure, Language Specification v2.41.
- Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, 2004.

Glossar

Dieses Glossar beruht in Teilen auf: Glinz, M.: A Glossary of Requirements Engineering Terminology. Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 1.1, May 2011.

Ablauf	*siehe* ↑Kontrollfluss
Aggregation:	Spezielle Art einer ↑Assoziation um Teil/Ganzes-Beziehungen zu modellieren.
Akteur:	Eine Person oder ein technisches System im ↑Kontext eines Systems, welche(s) mit dem betrachteten System interagiert.
Aktion:	In der Anforderungsmodellierung eine aus ↑Anforderungssicht nicht mehr zerlegbare ↑Funktion des ↑Systems; eine primitive ↑Funktion.
Aktivität:	In der Anforderungsmodellierung eine aus Anforderungssicht komplexe ↑Funktion des System-under-Development, die in weitere ↑Aktivitäten oder ↑Aktionen zerlegt werden kann.
Aktivitätsdiagramm:	Ein UML-Diagramtyp, der den Ablauf von ↑Aktionen in einem ↑System oder einem Teilsystem modelliert, evtl. mit ↑Datenflüssen oder Verantwortungsbereichen ergänzt, wo es notwendig ist.
Alternativszenario:	Ein ↑Szenario, welches, bezogen auf ein spezifisches Resultat (z.B. einen spezifischen Mehrwert), die in Bezug auf ein ↑Hauptszenario eine alternative Sequenz von ↑Interaktionen zur Erreichung des fachlichen Mehrwerts beschreibt.
Anforderungsmodell:	Ein ↑Modell, welches zum Zwecke der Spezifikation von ↑Anforderungen konstruiert ist. Setzt sich aus ↑Diagrammen der verschiedenen ↑Anforderungssichten und textuellen Ergänzungen zusammen.
Anforderungssicht	Definiert aus Gründen der Komplexitätsbeherrschung eine spezifische Abstraktion auf die ↑Anforderungen eines ↑Systems, in der nur gewisse Sachverhalte betrachtet werden (z.B. ↑Zustände und ↑Zustandsübergänge des System-under-Development) und andere Sachverhalte bewusst nicht betrachtet werden. Typischerweise sind die verschiedene Anforderungssichten zu einem Gesamtmodell der Anforderungen zusammenführbar.
Assoziation:	Eine Beziehung zwischen Modellelementen, z.B. eine Beziehung zwischen↑Klassen in einem ↑Klassendiagramm.

Attribut:	Eine charakteristische Eigenschaft einer Entität oder eines Objekts. Attribute werden auf Typebene, d.h. in den Entitätstypen (ER-Diagramme) oder Klassen (Klassendiagramme) definiert.
Ausnahmeszenario:	Ein ↑Szenario, welches eine Sequenz von ↑Interaktionen beschreibt, die durchlaufen werden müssen, wenn ein definiertes Ausnahmeereignis im Betrieb des ↑Systems eingetreten ist. Im Requirements Engineering werden Ausnahmeszenarien zur kontrollierten Behandlung von ↑Szenarien häufig komplementär zu ↑Hauptszenarien und/oder ↑Alternativszenarien spezifiziert.
Datatype:	Spezifikation einer komplexen Datenstrukturen für die Definition von ↑Attributen.
Datenfluss:	Darstellung von Informationen (in einem ↑Datenflussdiagramm oder ↑Aktivitätsdiagramm), die zwischen ↑Systemkontext und/oder ↑Funktionen des ↑Systems ausgetauscht werden. (Daten in Bewegung, Ein- und Ausgaben von ↑Funktionen).
Datenflussdiagramm:	Ein Diagramm zur Modellierung der Systemfunktionalität (oder der Funktionalität einer Systemkomponente) durch Prozesse (auch Aktivitäten genannt), Datenspeicher und Datenflüsse. Eingehende Datenflüsse triggern Prozesse, welche die eingehenden Daten konsumieren, verarbeiten, persistente Daten in Datenspeichern lesen und schreiben sowie neue Datenflüsse produzieren, die als Zwischenergebnisse weitere Prozesse triggern oder als Endergebnisse das System verlassen.
Diagramm:	Grafische Beschreibung einer zusammenhängenden Menge von Eigenschaften des Betrachtungsgegenstandes. Instanz eines spezifischen ↑Diagrammtyps.
Diagrammtyp:	Definiert eine Klasse „gleichartiger“ ↑Diagramme und wird durch eine ↑Modellierungssprache definiert.
Ereignis:	Zeitloses Ereignis, welches das Eintreten einer Bedingung, die Terminierung einer ↑Aktion oder ↑Aktivität, oder das eintreffen eines ↑Datenflusses oder einer Nachricht charakterisiert.
Funktion (eines Systems)	In der Anforderungsmodellierung ein Oberbegriff für ↑Use-Cases, ↑Aktivitäten oder ↑Aktionen, die in einer Anforderungsspezifikation für das ↑System gefordert werden.
Generalisierung:	Ein Konzept zur Abstraktion über gemeinsame Eigenschaften von z.B. ↑Klassen, in dem die gemeinsamen Eigenschaften in einem generalisierten Konzept (zusammengeführt und die Unterschiede in jeweils spezialisierten Konzepten abgebildet werden.

Hauptszenario:	Ein ↑Szenario, welches, bezogen auf ein spezifisches Resultat (z.B. einen spezifischen Mehrwert), die überwiegend auftretende Sequenz von Interaktionen zur Erreichung dieses Resultates beschreibt.
Instanzszenario:	Ein ↑Szenario, in welchem Kommunikationspartner und Interaktionen auf der Instanzebene betrachtet werden.
Interaktion:	Eine Interaktion zwischen Kommunikationspartnern ist ein Fluss von materiellen (z.B. Geld) oder immateriellen Dingen (z.B. Information) zwischen zwei oder mehr Kommunikationspartnern.
Interaktionsbezogene Sicht:	Die interaktionsbezogene Sicht ist eine spezielle ↑dynamische Sicht auf die ↑Anforderungen des ↑System-under-Development, bei der das Verhalten durch Interaktionen zwischen Kommunikationspartnern betrachtet wird.
Klasse	Repräsentiert eine Menge von ↑Objekten derselben Art durch die Beschreibung der Stuktur der Objekte und der Art und Weise, wie diese manipuliert werden können sowie deren Verhalten.
Klassendiagramm	Eine diagrammatische Repräsentation eines Klassenmodells oder eines Teils eines Klassenmodells.
Kommunikationsdiagramm:	Ein Diagramm zur Modellierung des Verhaltens in der Interaktionsbezogenen ↑Sicht, welches eine sachlogisch zusammenhängende Menge von ↑Interaktionen zwischen Objekten und/oder Kommunikationspartnern betrachtet und dabei auf die Visualisierung der bilateralen ↑Interaktionen zwischen Kommunikationspartnern fokussiert. Die kausale Reihenfolge der ↑Interaktionen wird hierbei durch Sequenznummern angegeben.
Komposition:	Spezielle Art einer ↑Assoziation um Teil/Ganzes-Beziehungen zu modellieren.
Kontextdiagramm:	Eine diagrammatische Darstellung der ↑Kontextsicht. In der Strukturierten Analyse ist das ↑Kontextdiagramm die Wurzel einer Hierarchie von Datenflussdiagrammen.
Kontextsicht	Eine ↑Anforderungssicht, die sich auf die Abgrenzung der ↑Systemgrenze vom ↑Kontext konzentriert, d.h. auf die Betrachtung der ↑Akteure bzw. Nachbarsysteme des ↑System-under-Development und auf die Schnittstellen zwischen dem System und diesen Nachbarsystemen. Häufig wird in der Kontextsicht lediglich der ↑Operationelle Kontext des System-under-Development durch ↑Kontextdiagramme modelliert.
Kontrollfluss	Zeitliche bzw. logische Abfolge von z.B. ↑Funktionen, ↑Aktionen oder ↑Aktivitäten.

Modell:	Abstrahierendes Abbild einer existierenden oder Vorbild für eine geplante Realität (z.B. ein System).
Modellelement:	Ein atomarer Bestandteil eines Diagramms oder eine textuelle Ergänzung im Anforderungsmodell. Ein Modellelement repräsentiert typischerweise eine einzelne Anforderung an das System.
Modellierungskonstrukt:	Ein atomarer Bestandteil eines ↑Diagrammtyps (z.B. Klasse, Assoziation, Zustand oder Zustandsübergang).
Modellierungssprache:	Sprache zur Konstruktion von ↑Diagrammen eines spezifischen ↑Diagrammtyps.
Objekt:	Eine Ausprägung/Instanz einer ↑Klasse.
Operationeller Kontext	Der Teil des ↑Systemkontexts mit dem das ↑System im Betrieb in einer funktionalen Wechselwirkung steht, z.B. Benutzer, andere Systeme, technische oder physikalische Prozesse oder Geschäftsprozesse.
Pragmatik:	Teil der Definition einer ↑Modellierungssprache, die deren Verwendungszweck beschreibt und ggf. auch beschreibt, für welche spezifischen Zwecke in welcher Weise abstrahiert werden muss, um den Verwendungszweck möglichst gut zu erfüllen.
Pragmatische Qualität:	Umfang, in dem ein ↑Diagramm/↑Modell im Hinblick auf die Eignung der gewählten Abstraktion seinen Verwendungszweck erfüllt.
Rolle:	Bezeichnung einer ↑Klasse aus Sicht der anderen ↑Klasse für eine↑Assoziation.
Semantik:	Teil der Definition einer ↑Modellierungssprache, der für die ↑Notationselemente deren generelle Bedeutung festlegt (d.h. generell → Was ist die Bedeutung einer ↑Klasse in ↑Klassendiagrammen?; nicht →Was ist die Bedeutung der ↑Klasse „Kunde“ im ↑Klassendiagramm?).
Semantische Qualität:	Umfang, in dem ein ↑Diagramm/↑Modell die spezifische Sicht auf den Betrachtungsgegenstand korrekt und vollständig abbildet.
Sequenzdiagramm:	Ein Diagramm zur Modellierung des Verhaltens in der Interaktionsbezogenen ↑Sicht, welches eine sachlogisch zusammenhängende Menge von ↑Interaktion zwischen ↑Objekten und/oder Kommunikationspartnern und dabei auf die Visualisierung der möglichen kausalen Reihenfolgen der Interaktionen fokussiert.

Sicht:	Eine abstrahierte Repräsentation des betrachteten ↑Systems, die sich durch eines oder mehrere ↑Diagramme (mit textuellen Ergänzungen) bildet. Sichten können disjunkt oder überlappend sein. Bewusste Überlappungen nutzen wir zur Qualitätssicherung der Modelle (Konsistenz herstellen bei Betrachtung aus mehreren Perspektiven).
Signal:	Ein ↑Ereignis in oder außerhalb des ↑Systems, das für das betrachtete ↑System relevant ist.
Statechart:	*siehe* ↑Zustandsmaschine
Steuerfluss:	*siehe* ↑Kontrollfluss
Syntaktische Qualität:	Umfang, in dem das ↑Diagramm/↑Modell den zugrundeliegenden syntaktischen Vorgaben genügt.
Syntax:	Teil der Definition einer ↑Modellierungssprache, der die in der Modellierungssprache zur Verfügung stehenden ↑Notationselemente und deren Kombinierbarkeit (Grammatik) festlegt.
System	Gegenstand/Objekt mit definierten Grenzen und einer Schnittstelle über die der Gegenstand/das Objekt mit seiner Umgebung (Kontext) wechselwirkt. Besteht typischerweise aus einer Menge miteinander in Beziehung stehender Bestandteile.
Systemgrenze:	Scope des ↑Systems. Grenzt das ↑System von dessen Kontext ab (z.B. durch Verantwortlichkeiten und Ausschlüsse).
Systemkontext:	Die für die Definition der ↑Anforderungen an ein ↑System relevanten Aspekte außerhalb des Systems und deren Beziehungen untereinander und zu dem betrachteten System. Der Systemkontext umfasst den ↑Operationellen Kontext, d.h. der Teil der Umgebung mit dem das ↑System im Betrieb in einer funktionalen Wechselwirkung steht.
Systemumgebung	*siehe* ↑Operationeller Kontext
System-under-Development	Das im Rahmen des Requirements Engineering bzw. der Anforderungsmodellierung betrachtete ↑System. Gegenstand der Kontextbildung.
System-under-Study	Ein im Rahmen einer Systemanalyse betrachtetes bzw. zu analysierendes ↑System. Nicht notwendigerweise auch der Entwicklungsgegenstand.

Szenario:	Eine ↑Interaktion zwischen Kommunikationspartnern (häufig zwischen dem betrachteten ↑System und ↑Akteuren im ↑Systemkontext), die zu einem gewünschten (oder ggf. ungewünschten) Resultat führt. Im Requirements Engineering wird häufig der Mehrwert für einen ↑Akteur im ↑Systemkontext als wesentliches Resultat eines ↑Szenarios gesehen.
Transition:	Ein Übergang von einem ↑Zustand zu einem anderen, ausgelöst durch einen ↑Trigger.
Trigger:	Die Verarbeitung eines Signals als Auslöser einer ↑Transition.
Typszenario:	Ein ↑Szenario, in welchem Kommunikationspartner und Interaktionen(↑) auf der Typebene betrachtet werden. Szenarien(↑) innerhalb einer ↑Use-Case-Spezifikation sind häufig auf der Typebene, d.h. sie betrachten Typen von Kommunikationspartnern und Typen von ↑Interaktionen.
Use-Case:	Eine Beschreibung der möglichen Interaktion zwischen einem ↑Akteur und dem ↑System, die, wenn sie ausgeführt wird, einen Mehrwert erbringt.
Use-Case-Diagramm:	Ein Diagrammtyp der UML, der die Modellierung von ↑Akteuren und ↑Use-Cases eines Systems erlaubt. Die Grenze zwischen Akteur und Use-Cases stellt die ↑Systemgrenze dar.
Use-Case-Spezifikation:	Die textuelle Beschreibung eines ↑Use-Cases.
Use-Case-Szenario:	Eine mögliche Abfolge (Trace) der Interaktionen innerhalb eines ↑Use-Cases. Die möglichen Abfolgen werden durch die ↑Haupt-, ↑Alternativ- und ↑Ausnahmenszenarien eines ↑Use-Cases bestimmt.
Zustand:	Ein Zustand ist eine Zusammenfassung von bestimmten Bedingungen, die während eines Zeitintervalls für ein ↑System oder ein Teilsystem gelten.
Zustandsautomat:	Ein Zustandsautomat beschreibt durch eine Zusammenfassung von ↑Zuständen mit ↑Transitionen zwischen diesen das Verhalten oder einen Teil des Verhaltens eines Betrachtungsgegenstandes (z.B. eines ↑Akteurs, einer ↑Funktion, eines ↑Use-Case, oder des ↑Systems).
Zustandsdiagramm:	Die graphische Repräsentation eines ↑Zustandsautomaten.
Zustandsmaschine:	*siehe* ↑Zustandsautomat
Zustandsmaschinendiagramm	*siehe* ↑Zustandsdiagramme

Abkürzungsverzeichnis

AD	Aktivitätsdiagramm
BPMN	Business Process Model and Notation
CM	Kommunikationsdiagramm
CPRE	Certified Professional for Requirements Engineering
CRM	Customer Relationship Management
DFD	Datenflussdiagramm
EPK	Ereignisgesteuerte Prozesskette
ER	Entity Relationship
FMC	Fundamental Modeling Concepts
IREB	International Requirements Engineering Board
ISO	International Organization for Standardization
IT	Informationstechnologie
ITU	International Telecommunication Union
OMG	Object Management Group
RE	Requirements Engineering
SA	Strukturierte Analyse
SD	Sequenzdiagramm
SuD	System-under-Development
SuS	System-under-Study
SysML	System Modeling Language
UML	Unified Modeling Language

Literaturverzeichnis

[Balz2011] Balzert, H.: Lehrbuch der Objektmodellierung - Analyse und Entwurf mit der UML 2, Spektrum Akademischer Verlag, Heidelberg 2011.

[BDH+2012] Broy, M.; Damm, W.; Henkler, S.; Pohl, K.; Vogelsang, A.; Weyer, T.: Introduction to the SPES Modeling Framework. In: Pohl, K.; Hönninger, H.; Achatz, R.; Broy, M.: Model-based Engineering of Embedded Systems, Springer, Heidelberg 2012.

[Caro1995] Carroll, J. M.: The Scenario Perspective on System Development. In: J. M. Caroll (Hrsg.): Scenario-Based Design – Envisioning Work and Technology in System Development, Wiley, New York, 1995, S. 1-17.

[Chen1976] Chen, P.: The entity-relationship model: Towards a unified view of data, ACM Transactions on Database Systems, 1976.

[CoNM1996] Coad, P.; D. North, D.; Mayfield, M.: Object Models: Strategies, Patterns, and Applications, Prentice Hall, 1996.

[Cock2000] Cockburn, A.: Writing effective Use-Cases. Addison-Wesley Longman, Amsterdam 2000.

[Cohn2002] Cohn, M.: User Stories Applied: For Agile Software Development, Addison Wesley, 2002.

[DaLF1993] Dardenne, A.; Van Lamsweerde, A.; Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming, Vol. 20, Nr. 1-2, Elsevier Science, Amsterdam, 1993, S. 3-50.

[DaTW2012] Daun, M.; Tenbergen, B.; Weyer, T.: Requirements Viewpoint. In: Pohl, K.; Hönninger, H.; Achatz, R.; Broy, M.: Model-based Engineering of Embedded Systems, Springer, Heidelberg 2012.

[Davi1993] Davis, A. M.: Software Requirements – Objects, Functions, States. 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey 1993.

[DeMa1979] DeMarco, T.: Structured Analysis and System Specification, Yourdon Press, Prentice Hall, 1979

[Fowl1996] Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading, MA 1996.

[GaJV1996] Gamma, E.; Helm, R.;Johnson, R.; Vlissides, J.: Entwurfsmuster. 5 Auflage. Addison-Wesley, Reading, MA 1996.

[GaSa1977] Gane, C.; Sarson, T.: Structured Systems Analysis – Tools & Techniques. Improved System Technologies, New York 1977.

[Glin2011] Glinz, M.: A Glossary of Requirements Engineering Terminology. Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 1.1, May 2011.

[HaCa1993] Hammer, M., Champy, J.: Reengineering the Corporation: A Manifesto for Business Revolution, Harper Business Essentials, 1993.

[HaHP2001] Hatley, D., Hruschka, P., Pirbhai, I.: A Process for System Architeture and Requirements Engineering, Dorset House, 2001.

[Hare1987] Harel, D.: Statecharts – A Visual Formalism for Complex Systems. Science of Computer Programming, Vol. 8, Nr. 3, 1987, S. 231-274.

[HKDW2012] Hilbrich, R.; Van Kampenhout, J. R.; Daun, M.; Weyer, T.: Modeling Quality Aspects: Real-Time. In: Pohl, K.; Hönninger, H.; Achatz, R.; Broy, M.: Model-based Engineering of Embedded Systems, Springer, Heidelberg 2012.

[Hoff1998] Hoffmann, J.: MATLAB und Simulink – Beispielorientierte Einführung in die Simulation dynamischer Systeme. Additions-Wesley, Reading, MA 1998.

[Hrus2014] Hruschka, P.: Business Analysis und Requirements Engineering – Prozesse und Produkte nachhaltig verbessern. Carl Hanser Verlag, München 2014.

[IEEE1471] IEEE Recommended Practice for Architectural Description of Software Intensive Systems. IEEE Standard 1471-2000.

[ISO25010] ISO/IEC/IEEE Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation. ISO/IEC/IEEE Standard 25010:2011.

[ISO26702] ISO/IEC/IEEE Systems and Software Engineering – Application and Management of the Systems Engineering Process. ISO/IEC/IEEE Standard 26702:2005.

[ISO29148] ISO/IEC/IEEE Systems and Software Engineering – Life Cycle Processes – Requirements Engineering. ISO/IEC/IEEE Standard 29148:2011.

[ISO42010] ISO/IEC/IEEE Systems and Software Engineering – Architecture description. ISO/IEC/IEEE Standard 42010:2011.

[ITU2004] International Telecommunication Union: ITU-T Z.120 Message Sequence Chart (MSC), 2004.

[JCJO1992] Jacobson, I.; Christerson, M.; Jonsson, P.; Oevergaard, G.: Object Oriented Software Engineering – A Use-Case Driven Approach. Addison-Wesley, Reading, 1992.

[LaSi1987] Larkin, J. H.; Simon, H. A.: Why a diagramm is (sometimes) worth ten thousand words. In: Cognitive Science, Vol. 11, S. 65-99.

[LiSS1997] Lindland, O. I.; Sindre, G.; Sølverg, A.: Understanding Quality in Conceptual Modeling. IEEE Software, Vol. 22, Nr. 2, IEEE Press, 1994, 42-49.

[Mart1989] Martin, J.: Information Engineering, Book I – Introduction. Prentice Hall, Englewood Cliffs 1989.

[McPa1984] McMenamin, S. M.; Palmer, J. F.: Essential Systems Analysis. Prentice Hall, London 1984.

[Nuse2001] Nuseibeh, B.: Weaving Together Requirements and Architectures. IEEE Computer, Vol. 34, Nr. 3, IEEE Computer Society, Los Alamitos, 2001, S. 115-117.

[OMG2012] OMG Object Constraint Language (OCL); Version 2.3.1; January 2012 http://www.omg.org/spec/OCL/2.3.1.

[OMG2010a] Object Management Group: OMG Systems Modeling Language (OMG SysML) Language Specification v1.2. OMG Document Number: formal/2010-06-02.

[OMG2010b] Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure, Language Specification v2.41.

[OMG2010c] Object Management Group: OMG Unified Modeling Language (OMG UML), Infrastructure, Language Specification v2.41.

[OMG2011] Object Management Group: OMG Business Process Model and Notation (OMG UML), Language Specification v2.0.

[Pohl2010] Pohl, K.: Requirements Engineering – Fundaments, Principles, Techniques. Springer, Heidelberg 2010.

[RuJB2004] Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual, Addison Wesley, Reading, MA 2004.

[PoRu2011] Pohl, K.; Rupp, C.: Basiswissen Requirements Engineering. 3. Auflage, dpunkt, Heidelberg 2011.

[Pott1995] Potts, C.: Using Schematic Scenarios to Understand User Needs. In: Proceedings of the ACM Symposium on Designing Interactive Systems – Processes, Practices, Methods and Techniques (DIS'95), ACM, New York, 1995, S. 247-266.

[RaJa2001] B. Ramesh, M. Jarke: Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, Vol. 27, No. 1, IEEE Press, 2001, S. 58-93.

[RiWe2007] Rinke, T.; Weyer, T.: Defining Reference Models for Modelling Qualities - How Requirements Engineering Techniques can Help. In: Proc. of the 13th Intl. Working Conf. on Requirements Engineering – Foundation for Software Quality, Lecture Notes in Computer Science, 4542, Springer 2007.

[RoRo2006] Robertson, S.; Robertson, J.: Mastering the Requirements Process. 2nd edition, Addison-Wesley, Amsterdam, 2006.

[RAC+1998] Rolland, C.; Achour, C.; Cauvet, C.; Ralyté, J.; Sutcliffe, A.; Maiden, N.; Jarke, M.; Haumer, P.; Pohl, K.; Dubois, E.; Heymans, P.: A Proposal for a Scenario Classification Framework. In: Requirements Engineering, 3 (1998) 1, S. 23-47.

[RoSc1977] Ross, D. T.; Schoman, K.E.: Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering, Vol. 3, Nr. 1, 1977, S. 6-15.

[Rupp2007] Rupp, C.: Requirements-Engineering und -Management – Professionelle, iterative Anforderungsanalyse für die Praxis. Carl Hanser Verlag, München 2007.

[RuQu2012] Rupp, C.; Queins, S.: UML 2 Glasklar – Praxiswissen für die UML-Modellierung. Carl Hanser Verlag, München 2012.

[Sche2001] Scheer, A.-W.: ARIS Modellierungsmethoden, Metamodelle, Anwendung. 4. Auflage, Springer, Berlin 2001.

[ShMe1988] Shlaer, S.; Mellor, S.: Object-oriented Systems Analysis – Modeling the World in Data. Prentice Hall, Englewood Cliffs 1988.

[Weye2010] Weyer, T.: Kohärenzprüfung von Verhaltensspezifikationen gegen spezifische Eigenschaften des operationellen Kontexts. Dissertationsschrift, Universität Duisburg-Essen, Fakultät für Wirtschaftswissenschaften, Essen 2010.

Index

A

Abbruch Interaktion 98
Abbruchfragment 98
Ablauflogik 9
Abstraktion 78
Abstraktionsebene 63
Abstraktionsmechanismus 61
Aggregation 35
Aktivierung 92
Aktivitäten
  bündeln 61
  zerlegen 61
Aktivitätsdiagramm 57, 64
Alternative Interaktion 95
Alternativszenario 60, 88
Änderungsprojekt 6
Anforderung 1, 6
Anforderungen
  modellieren 1
Anforderungsdiagramm 2, 11
Anforderungsmodell 11
  Qualität 13
Anforderungsmodellierung 2
Annahme 99
Architekturstil 6
Assoziation 32
  Abgeleitete 42
Asynchrone Nachricht 93
Attribut 26, 28
  Abgeleitetes 27
Ausnahmeereignis 50, 60
Ausnahmeszenario 60, 88

B

Balancing-Regel 63
Blockdigramm 18
Boolean 30
BPMN 4, 9, 54, 111
Bubble 56
Bündeln von Aktivitäten 61
Bündelung von Funktionen 61
Business-Analyse 4, 9

C

Character 30
Complete 41
Constraint 31, 37, 40
Constraints ..... *Siehe* Randbedingung
Constraints-Sicht 8

D

Datenfluss 54, 65
Datenflussdiagramm 54, 56
Datenspeicher 56, 63, 66
Datentyp 26, 30
Default Entry 77
Defer 71
Diagrammtyp 4
Dienstaufruf 93
Disjoint 41
Domänenglossar 10

E

Eingabeparameter 93
Eingebettetes System 4
Endlicher Automat 67
Endzustand 75
Entity-Relationship-Diagramm 21
Entwurf 6
Entwurfsdiagramm 6
Entwurfsentscheidung 6
Enumeration 29
Essenzmodell 5
Explicit Entry 77
Extend-Beziehung 52
Externe Auslöser 49

F

Fachlichen Daten 21
Flache Historie 77
Float 30
Formalisierungsgrad 13
Funktion 94
Funktionen
  bündeln 61

zerlegen 61
Funktionssicht 7
Funktionsspezifikation
Diagrammatische 54
Textuelle 62

G

Generalisierung 39, 52
Geschachtelte Interaktion 98
Gleichartigkeit 41
Glossar 21
Granularität 49, 50, 54
Guard 67, 73

H

Hauptszenario 60, 88
Heuristik
Ermittlung einfacher Beziehungen 34
Ermittlung von Aggregationen 35
Ermittlung von Attributen 27
Ermittlung von Datentypen 31
Ermittlung von Generalisierung 41
Ermittlung von Klassen 24
Historie 77
Flache 77
Tiefe 77

I

Include-Beziehung 52
Incomplete 40
Informationsstrukturmodellierung 21
Informationssystem 4
Inkarnation 5
Inkarnationsmodell 5
Integer 30
Interaktion
Abbruch 98
Alternative 95
Geschachtelt 98
Optionale 96
Referenzierte 96
Wiederholung 97
IREB Glossary 1, 3
Ist-Modell 5

K

Klasse 23, 28
Klassendiagramm 22, 49, 95
Kommunikationsdiagramm 89
Komposition 35
Kompositum 42
Konsistenzbedingung 63
Kontextdiagramm 17, 18, 19, 48, 49, 53, 57, 63, 92, 107
Kontextdiagramme
datenflussorientiert 19
Kontextmodellierung 20
Kontextsicht 17
Kontrollfluss 54, 61, 65, 92
Kontrollflussorientierte Sicht 9
Koordinator 42

L

Lebenslinie 92
Leserichtung 33, 38

M

Mapping 13
MATLAB 4
Mehrdeutigkeit 13
Mehrwert 87
Modellelement 11, 15
Modellelemente 3
grafisches 4
textuelles 4
Modellierungskonstrukt 3
textuell 4
Modellierungssprache
Anpassen 10
Modellierungswerkzeug 14
Multiplizität 26, 33

N

Nachbedingung 50
Nachricht 87, 100
Asynchron 93
Synchron 93
Nachrichtenaustausch 100
Name 26, 33
Navigierbarkeit 38

O

Object Contraint Language ..... *Siehe* OCL
Objekt 23
Objektfluss 58
OCL 32, 39

Optionale Interaktion 96
orthogonale Region 68
Orthogonalen Region 81
Overlapping 41

P

Paketieren 53
Party 42
Petri-Netz 55
Pin 59
Pragmatische Qualität 15
Primitiver Datentypen 29
Process 56

Q

Quelle 19, 57

R

Randbedingung 5
realize 12
Referenzierte Interaktion 96
refines 12
Requirements Engineering 1, 17
Rolle 33
Rollen 25
Rückgabeparameter 93

S

satisfies 12
Scope 17, 19, 47, 109
Semantik 11
Semantische Qualität 14
Senke 19, 57
Sequenzdiagramm 89
Sequenznummer 100
Signal 64, 74
Signaleingang 61
Signalempfang 64
SignalTrigger 73
Simulink 4
Sink ..... *Siehe* Senke
Soll-Modell 6
Source ..... *Siehe* Quelle
Startzustand 75
Statechart 67
Stateflow-Diagramm 4
Statisch-strukturelle Sicht 7
Stereotyp 10, 11, 64, 90
Steuerfluss ..... *Siehe* Kontrollfluss
String 30, 32
Strukturierte Analyse 54
Strukturierter Datentyp 29
Synchrone Nachricht 93
Syntaktische Qualität 14
Syntax 11
Systemanalyse 5
Systemarchitektur 6
Systemfunktion 94
Systemgrenze 17
Systemmodell 4
Systemvision 87
Szenariomodellierung 87
Szenariosicht 10, 89

T

Teile und Herrsche 13
Teilprozess 48
Terminator 19
Textuelle Funktionsspezifikation 62
textueller Ergänzung 4
Tiefe Historie 77
TimeTrigger 73
Transition 67
  Selbst- 73
Trennung von Belangen 12
Trigger
  bei Transitionen 73
  in Zuständen 71
Twin-Peaks-Modell 6

U

UML-Werkzeug 11
Unterbrechnungsbereich 61
Unterbrechungsbereich 64
Unterzustand 78
Unterzustandsautomatenzustand 79
Use-Case 46, 85, 88
Use-Case-Diagramm 47, 48, 92
Use-Case-Modell 46
Use-Case-Schablone 50
Use-Case-Spezifikation 50

V

Verhalten
  bei Transitionen 73
  in Zuständen 71
Verhaltenssicht 7

Vorbedingung 50

W

Wiederholte Interaktion 97

Z

Zeitauslöser 49
Zerlegen von Aktivitäten 61
Zerlegung von Funktionen 61
Zustand 66
  aktiv 71
  Austrittsverhalten 71
  Eintrittsverhalten 71
  Hierarchisierung 76
  Unter- 76
  verlassen 73
Zustandsautomat 66
Zustandsmaschinendiagramm 69
Zustandsorientierte Sicht 10
Zustandsverhalten 71
Zustandswechsel 94