Strukturierte Nutzungssicht für multifunktionale Systeme

Martin Deubler

Institut für Informatik
der Technischen Universität München

Strukturierte Nutzungssicht für multifunktionale Systeme

Martin R. Deubler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Hans Michael Gerndt
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2.	Univ.-Prof. Dr. Johann Schlichter

Die Dissertation wurde am 16. April 2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 27. August 2008 angenommen.

Für meine Familie

Kurzfassung

Moderne Softwaresysteme bieten den Anwendern eine große Anzahl unterschiedlichster Funktionen oder Dienste. Aber nicht nur der Trend zu immer mehr und neuen Funktionen ist ungebrochen, es bieten sich auch immer mehr Möglichkeiten des Zugangs zu den einzelnen Funktionen. Unabhängig von einer Anwendungsdomäne verwenden wir für solche Systeme den Begriff des „multifunktionalen Systems“. Typischerweise sind bei multifunktionalen Systemen die Funktionen nicht alle voneinander unabhängig. Für eine Funktion sind Mehrfachinteraktionen mit unterschiedlichen Beziehungen zu anderen Funktionen symptomatisch. Um nun bei einer immer größer werdenden Anzahl von Funktionen überhaupt noch verlässliche Aussagen über das Verhalten einer Funktion in Kombination mit anderen Funktionen oder letztendlich über das Systemverhalten treffen zu können, ist die Spezifikation der Nutzungsfunktionen samt deren Zusammenhänge unerlässlich. Eine derartige Beschreibung der Gesamtfunktionalität eines multifunktionalen Systems aus der Nutzerperspektive ist eine wesentliche Aufgabe des Requirements Engineerings. Nur so kann der oft immense Aufwand zur Behebung von ungewollten „Feature Interactions“ vermieden oder zumindest durch eine erhöhte Nachvollziehbarkeit des Verhaltens der Funktionen erheblich reduziert werden. Die Nutzungssicht eines multifunktionalen Systems mithilfe geeigneter Funktionsbeziehungen zu strukturieren ist Gegenstand dieser Arbeit.

In dieser Arbeit werden Dienste als zentrale Modellierungseinheiten des technischen Requirements Engineerings betrachtet. Sie repräsentieren dabei einerseits die funktionalen Anforderungen, die von den Nutzern des Systems als entsprechende Teilverhalten des Systems wahrgenommen werden. Wir verwenden daher auch den Begriff der „Nutzungsfunktion“. Auf der anderen Seite werden aber auch im Rahmen der Anforderungsanalyse aus den technischen Anforderungen heraus zusätzlich deren Beziehungen mit anderen Diensten identifiziert, und zwar unter der übergeordneten Fragestellung „wie wird das System genutzt?“. Hierbei dienen unterschiedliche Sichten, die sich auf hierarchische Zusammenhänge, aber insbesondere auch auf Verhaltensabhängigkeiten konzentrieren, der Strukturierung des Systems. Das Ergebnis ist ein aus den verschiedenen Perspektiven überlagertes Beziehungsmodell, die so genannte „Nutzungsarchitektur“ oder auch „logische Dienstarchitektur“.

Im Rahmen dieser Arbeit werden aktuelle Ansätze zur Beschreibung von Softwaresystemen untersucht und auf ihre Eignung hinsichtlich der Modellierung von Funktionsbeziehungen bewertet. Bei dem Gros aller Ansätze ist eine explizite Modellierung nicht vorgesehen. Sie erfolgt dort im Wesentlichen erst implizit während des Entwurfs durch die Spezifikation von Kommunikationsprotokollen. Oft erlaubt eine eingeengte Sicht auf das Systemmodell zudem nur die Identifizierung eines speziellen Zusammenhangs, wie etwa die *Nutzung* eines Diensts. Diese Arbeit geht darüber hinaus und untersucht Zusammenhänge im Kontext einer so genannten „Nutzungssicht“. Mit der vorgestellten Menge von

Beziehungstypen können Abhängigkeiten zwischen Funktionen praktikabel beschrieben werden.

In der Entwicklungsphase zwischen der Anforderungsanalyse und dem Systementwurf hilft die Nutzungsarchitektur, die Interaktionen der Dienste zu bewerten und damit insbesondere auch potenziell unkontrollierte Seiteneffekte frühzeitig mit der Festlegung von geeigneten Beziehungen zu verhindern. Es ist nun möglich, Aussagen über Dienste, beziehungsweise über das Zusammenspiel von Funktionen in Kombination, treffen zu können, ohne eine exakte Spezifikation, eine Schnittstellenbeschreibung oder eine innere Struktur (Glassboxsicht) kennen zu müssen. Mit der Nutzungsarchitektur eines Systems und den zugrunde liegenden Funktionsbeziehungen wird gewissermaßen eine Strukturierung der Blackboxsicht erreicht.

Das Ergebnis der Arbeit ist eine überschaubare und praktikable Menge von Beziehungstypen, die zur Beschreibung von Zusammenhängen zwischen Nutzungsfunktionen ausreicht. Die Charakterisierung der Beziehungstypen stützt sich auf formale Grundlagen, insbesondere auf Konzepte der theoretischen Modelle FOCUS und JANUS, ab. Beziehungen zwischen Funktionen können somit explizit beschrieben werden, was den Rahmen eines modellbasierten Requirements Engineerings konsequent erweitert. Konsequenzen für Spezifikationen, wie die Definition geeigneter Interaktionsschnittstellen, können daraus unmittelbar abgeleitet werden. Eine Fallstudie anhand ausgewählter Funktionen eines Mobiltelefons verdeutlicht die vorgestellten Konzepte.

Danksagung

Die Zeit, die ich am Lehrstuhl für Software & Systems Engineering als wissenschaftlicher Mitarbeiter verbringen durfte, gehört fraglos zu den angenehmsten und spannendsten Abschnitten meiner beruflichen Laufbahn. Nicht zuletzt wurde angesichts der kollegialen und produktiven Atmosphäre auch dort der Grundstein für die Entstehung dieser Dissertation gelegt. Dafür möchte ich Professor Manfred Broy sehr herzlich danken. Insbesondere bedanke ich mich aber auch dafür, dass er sich darauf einließ, mich – aus der Industrie kommend – als Mitarbeiter seines Lehrstuhls aufzunehmen, obwohl oder gerade weil doch bereits einige Jahre seit meinem Diplomabschluss vergangen waren.

Darüber hinaus möchte ich mich für die Geduld und Unterstützung, die mir Professor Manfred Broy bei der Erstellung meiner Arbeit entgegengebracht hat, bedanken. Sowohl unsere Diskussionen als auch seine wertvollen Anmerkungen habe ich stets sehr geschätzt. Mein Dank gebührt auch Professor Johann Schlichter für die Übernahme des Zweitgutachtens und sein Verständnis für meine außergewöhnliche Situation.

Die Umsetzung des Entschlusses, nach vielen Jahren in der Industrie zurück an die „technische Front“ zu wechseln, wäre nicht möglich gewesen, wenn es meine damaligen Vorgesetzten bei der Siemens AG nicht ermöglicht, ja unterstützt hätten. Dafür bin ich Dr. Wolfgang Böhm und Dr. Eduard Scheiterer zu großem Dank verpflichtet.

Nach drei Jahren wissenschaftlichen Arbeitens kehrte ich an meine alte Wirkungsstätte bei der Siemens AG beziehungsweise der heutigen Nokia Siemens Networks GmbH zurück. Mein besonderer Dank gilt hier ebenfalls Dr. Eduard Scheiterer, ohne dessen großes Entgegenkommen die Vollendung dieser Arbeit kaum möglich gewesen wäre.

Die erfolgreiche und interessante Zusammenarbeit mit allen ehemaligen Lehrstuhlkollegen in den Forschungsprojekten WEIT und MEWADIS war Ausgangspunkt meiner Arbeit und Inspiration zugleich. Dafür sei Professor Andreas Rausch, Dr. Michael Gnatz und Michael Meisinger, sowie Dr. Gerhard Popp, Dr. Guido Wimmel und Johannes Grünbauer, aber auch alle anderen Mitarbeitern des Lehrstuhls gedankt. Für das aufmerksame Durchlesen und die Kommentare zu Vorversionen dieser Arbeit möchte ich Johannes Grünbauer besonders danken. Frau Silke Müller danke ich insbesondere für ihre Hilfsbereitschaft und Ansprechbarkeit in allen organisatorischen Angelegenheiten auch außerhalb meines Engagements bei der Technischen Universität München.

Schließlich wäre ohne die enorme Geduld und Rücksicht meiner Familie während der Erstellung meiner Arbeit, insbesondere in den letzten beiden Jahren mit großen Entbehrungen an Wochenenden und Ferienzeiten, ein Abschluss undenkbar gewesen. Die liebevolle Unterstützung meiner Frau Petra und meiner Tochter Mona haben mich aufrecht gehalten. Danke!

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Problembeschreibung . . . 3
- 1.2 Begriffsdefinitionen . . . 8
- 1.3 Ansatz und Ergebnisse . . . 11
- 1.4 Inhalt und Aufbau . . . 13

2 Verwandte Arbeiten **15**
- 2.1 Konzepte des Requirements Engineerings . . . 16
- 2.2 Architekturen . . . 25
- 2.3 Dienstbasierte Modellierung . . . 28
- 2.4 Konfigurationen und Varianten . . . 34
- 2.5 Plattformen . . . 35
- 2.6 Zusammenfassung . . . 38

3 Grundlagen **43**
- 3.1 Nachrichtenströme, Kanäle und Kanalhistorien . . . 43
- 3.2 Operatoren und Relationen . . . 46
- 3.3 Kombination von Funktionen . . . 52
- 3.4 Notationen . . . 57

4 Grundlegende Funktionsbeziehungen **61**
- 4.1 Die Nutzungssicht auf ein System . . . 62
- 4.2 Syntaktische Zusammenhänge von Funktionen . . . 66
- 4.3 Hierarchie von Funktionen . . . 67
- 4.4 Einbettung von Funktionen in ein Supersystem . . . 69
- 4.5 Kombinierbarkeit von Funktionen . . . 89
- 4.6 Unabhängigkeit von Funktionen . . . 93

5 Das Verhalten kombinierter Funktionen 99
5.1 Nicht kombinierbare Funktionen 102
5.2 Kontrollierter Zugang zu Funktionen 108
5.3 Varianten und Muster von Einflussnahmen 116
5.4 Zusätzliche Verhaltensmuster in Kombination 125
5.5 Zusammenfassung 128

6 Fallstudie 133
6.1 Die Funktionen im Überblick 134
6.2 Spezifikationen als Zustandsautomaten mit Ein- und Ausgabe 138
6.3 Von isolierten zu kombinierten Funktionsspezifikationen 139
6.4 Weitere Funktionen und Beziehungen 154
6.5 Unabhängige Funktionen und implizite Beziehungen 173
6.6 Die Nutzungsarchitektur der Mobiltelefonfunktionen 174

7 Zusammenfassung und Ausblick 179

Literaturverzeichnis 185

Definitionsverzeichnis

1.1 Funktionale Anforderung 8
1.2 Dienst, engl. Service 9
1.3 Feature Interaction 10
1.4 Blackboxsicht 10
1.5 Nutzungsarchitektur 11
3.1 Schnittmenge von Kanalmengen 46
3.2 Vereinigungsmenge von Kanalmengen 46
3.3 Subtyp einer Kanalmenge 47
3.4 Subtyp einer Schnittstelle 47
3.5 Erste und letzte Nachricht eines Nachrichtenstroms 48
3.6 Erste und letzte Nachricht einer Kanalhistorie 48
3.7 Präfix einer Kanalhistorie 49
3.8 Überlappung von Nachrichten in Sequenzen 49
3.9 Überlappung von Nachrichten in Strömen 50
3.10 Überlappung von Nachrichten in Kanalhistorien 51
3.11 Verzögerte Nachrichten in Kanalhistorien 51
3.12 Domäne einer Funktion 53
3.13 Wertebereich einer Funktion 53
3.14 Projektion einer Historie auf getypte Kanäle 53

3.15 Verhaltensverfeinerung . . . 55
4.1 Der Beziehungstyp subfunction . . . 68
4.2 P_0 – Die Menge der undefinierten Eingaben . . . 76
4.3 P_V– Die Menge der gültigen Eingaben . . . 77
4.4 P_S – die Menge der vorgabegemäßen Eingaben . . . 78
4.5 P_D – die Menge der abweichenden Eingaben . . . 79
4.6 P_N– Die Menge der neutralen Eingaben . . . 79
4.7 P_A– Die Menge der beeinflussenden Eingaben . . . 79
4.8 Die Kanalmenge I_A aller beeinflussenden Nachrichten . . . 80
4.9 Kombinierbarkeit von Funktionen . . . 90
4.10 Der Beziehungstyp independent . . . 94
4.11 Wechselseitige Unabhängigkeit von Funktionen . . . 94
5.1 Der Beziehungstyp affect . . . 101
5.2 Der Beziehungstyp exclude . . . 103
5.3 Der Beziehungstyp prepare . . . 105
5.4 Der Beziehungstyp replace . . . 106
5.5 Der Beziehungstyp refine . . . 107
5.6 P_{ENA}– Enable Set . . . 109
5.7 Der Beziehungstyp enable . . . 110
5.8 P_{DIS}– Disable Set . . . 112
5.9 Der Beziehungstyp disable . . . 112
5.10 Der Beziehungstyp direct . . . 114
5.11 Der Beziehungstyp cancel . . . 117
5.12 Der Beziehungstyp interrupt . . . 119
5.13 Der Beziehungstyp trigger . . . 121
5.14 Der Beziehungstyp consult . . . 123
5.15 Der Beziehungstyp extend . . . 126
5.16 Der Beziehungstyp complement . . . 127

Abbildungsverzeichnis

1.1 Lücke im Requirements Engineering? 4
1.2 Gegensätzliche Anforderungen 6
1.3 Produktmodell 7
1.4 Modellierung von Funktionsbeziehungen als Übergang zum Systemdesign . 12

3.1 Syntaktische Schnittstelle 45
3.2 Transition eines Zustandsautomaten mit Eingabe und Ausgabe 58

4.1 Perspektiven einer Nutzungsarchitektur 63
4.2 Strukturierte Blackboxsicht 64
4.3 Syntaktische Zusammenhänge zweier Nutzungsfunktionen 66
4.4 Beziehungsmuster kombinierbarer Funktionen 70
4.5 Kontext einer betrachteten Funktion g – ein Supersystem s 72
4.6 Restriktion der Ausgabekanäle auf O_g 73
4.7 Die Kanäle aller beeinflussenden Nachrichten von g 80
4.8 Inkonsistente Spezifikation kombinierter Funktionen 89

5.1 Schema einer replace Beziehung im Zustandsübergangsdiagramm 106
5.2 Schema einer refine Beziehung im Zustandsübergangsdiagramm 108
5.3 Schema einer enable Beziehung im Zustandsübergangsdiagramm 111
5.4 Schema einer direct Beziehung im Zustandsübergangsdiagramm 115
5.5 Spezialfall einer direct Beziehung 115
5.6 Schema einer cancel Beziehung im Zustandsübergangsdiagramm 118
5.7 Schema einer interrupt Beziehung im Zustandsübergangsdiagramm 120
5.8 Schema einer trigger Beziehung im Zustandsübergangsdiagramm 122
5.9 Einschalten eines Mobiltelefons mit «PIN Eingabe» 124
5.10 Schema einer extend Beziehung im Zustandsübergangsdiagramm 126

5.11 Schema einer complement Beziehung im Zustandsübergangsdiagramm . . . 128
5.12 Beziehungstypen im Überblick 131

6.1 Ausgewählte Nutzungsfunktionen eines Mobiltelefons 134
6.2 Abstrakte Nachrichten mit beispielhaften Repräsentationen im Mobiltelefon 139
6.3 Einschalten des Mobiltelefons mit Einschaltsicherung (isolierte Spezifikation) 141
6.4 Ausschalten des Mobiltelefons (isolierte Spezifikation) 141
6.5 Ein- und Ausschalten des Mobiltelefons mit Einschaltsicherung 143
6.6 Einschalten des Mobiltelefons mit Zugriffsschutz (isolierte Spezifikation) . 143
6.7 Einschalten mit Einschaltsicherung oder Zugriffsschutz 145
6.8 PIN eingeben mit den Subfunktionen «Korrekte PIN eingeben», «Falsche PIN eingeben», sowie «PIN Eingabe abbrechen» 146
6.9 PIN Kontrolle (Aktivierung und Deaktivierung) mit PIN Prüfung 149
6.10 Ein- und Ausschalten des Mobiltelefons mit Einschaltsicherung beziehungsweise mit Zugriffsschutz 150
6.11 Navigieren 153
6.12 Telefonieren mit den Subfunktionen «Anruf annehmen», «Anruf abweisen», «Anruf beenden», «Nummer anrufen» mit internationaler Vorwahl sowie «Nummer wählen» 155
6.13 Adressen mit den Subfunktionen «Eintrag anlegen», «Eintrag entfernen» und «Namen anrufen» 159
6.14 Anruf mit mehrfacher Signalisierung annehmen 161
6.15 Betriebszustand anzeigen 162
6.16 Alarm „Bundle“ 163
6.17 Letzte Nummer wiederholen 166
6.18 Wahl wiederholen und gewählte Nummern löschen (Ruflisten „Bundle“) . . 167
6.19 Jede Taste 169
6.20 Tastensperre mit den Subfunktionen «Tasten sperren» und «Tasten entsperren» 170
6.21 Anrufannahme in Kombination mit «Jede Taste» und «Tastensperre» . . . 172
6.22 Nutzungsarchitektur der ausgewählten Nutzungsfunktionen eines Mobiltelefons 175
6.23 Beziehungen eines bestimmten Typs, hier prepare 176
6.24 Beziehungskontext einer Nutzungsfunktion, hier «Wecker» 177

Kapitel 1

Einführung

Der Trend in der Informationstechnologie zu immer größerer Rechenleistung und mehr Funktionalität, die durch Software realisiert wird, ist ungebrochen. Dies führt zu einer immer größeren Anzahl an verwendeten Funktionen [Bro04b], aber insbesondere auch an logischen Abhängigkeiten. Die Komplexität eines Systems wird dadurch immer höher [Bro04a] und die Nachvollziehbarkeit des Systemverhaltens wird in der Regel immer schwieriger.

„Sind diese Systemfunktionen unabhängig voneinander?“, „Kann diese Systemfunktion erweitert werden, ohne eine andere (negativ) zu beeinflussen?“ – die eindeutige Beantwortung dieser Fragen ist nicht nur beim Entwurf von Softwaresystemen von großem Interesse. Auch bei der Fehlersuche in einem laufenden Softwaresystem oder bei der Erweiterung um so genannte Features spielt das exakte Wissen um die Zusammenhänge und internen Beziehungen der Funktionalitäten des Systems eine entscheidende Rolle. Doch sind diese Fragen so einfach zu beantworten oder sind die Antworten gar offensichtlich? Ist das Systemverständnis unter Einbeziehung der klassischen – wenn vorhanden – fachlichen oder technischen Architekturen dafür ausreichend? Gerade bei multifunktionalen Softwaresystemen ist nach unserer Erfahrung selbst eine große Expertise eines Systemsoftwareentwicklers nicht ausreichend, alle Zusammenhänge und Seiteneffekte intuitiv zu erkennen.

Ein systematisches Anforderungsmanagement ist notwendig, aber nicht hinreichend für ein umfassendes Wissen über das Systemverhalten. Über geforderte Funktionalitäten hinaus scheint es noch – zunächst unbekannte – (Nutzungs-) Anforderungen zu geben, die im Nutzungskontext versteckt sind. Ein konsequentes modellbasiertes Requirements Engineering wird dazu beitragen, in einer so frühen Projektphase wie möglich alle, auch implizit geforderten Systemeigenschaften erfassbar und nachvollziehbar zu machen.

Die Stimmigkeit der Anforderungen an das System ist nicht zuletzt ausschlaggebend für die Qualität eines Produkts. Auch die Kosten werden entscheidend durch die Anforderungen bestimmt. Overengineering führt oft zu ungerechtfertigten Anforderungen und macht das Produkt unnötig teuer und komplex. Gerade die Entwicklung von multifunktionalen Systemen ist ohne eine saubere Anforderungsanalyse und Anforderungsdefinition kaum denkbar. Inkonsistenzen in den Anforderungen oder in den Datenmodellen werden aufgrund der komplexen Zusammenhänge nicht frühzeitig erkannt. Deren (ungewollte) Auswirkungen entsprechen dann in der Regel nicht dem gewünschten Systemverhalten (aus Nutzungssicht) und stellen somit einen erheblichen Qualitätsmangel dar. Im schlechtesten Fall werden solche Fehler erst im Feld entdeckt, deren Behebung – wenn überhaupt möglich – sehr kostenintensiv ist.

Da ein Softwaresystem in erster Linie durch sein Verhalten charakterisiert ist, ist die konzeptuelle Umsetzung von funktionalen Anforderungen durch die Partitionierung der Systemfunktionalität in Dienste – wie bei der dienstorientierten Modellierung – in der Regel vergleichsweise intuitiv. Mit einem Dienst wird ein Teilverhalten eines Systems beschrieben. Teilverhalten sind durch Interaktionen charakterisiert, denen so genannte Interaktionsmuster zugrunde liegen [KGM+04, Bro03a, Bro05]. Das Verhalten, insbesondere auch über die Systemgrenzen hinaus, ist im Vergleich zu rein komponentenbasierten Ansätzen leichter zu spezifizieren. Wenn nur einzelne Komponenten betrachtet werden, ist es sehr schwierig, solche Interaktionen überhaupt zu erfassen [KGM+04].

Multifunktionale Softwaresysteme stützen sich auf eine große Anzahl von Funktionen ab [DGP+04b, DGP+04a]. Ein fundamentaler Unterschied zu klassischen Anwendungssystemen ist der hohe Grad an Interaktionen und Abhängigkeiten der Funktionen. Die Beziehungen und Abhängigkeiten zwischen den Funktionen sind – wenn überhaupt – nur schwer zu überblicken. Die durch die Kombination unterschiedlicher Funktionen verursachten Auswirkungen auf das Systemverhalten sind kaum abzuschätzen. Aber selbst bei Kenntnis von genauen Spezifikationen aller Funktionen ist es nahezu unmöglich, ein Systemverständnis zu entwickeln. Eine gängige Methode, die immer komplexer werdenden Systeme überhaupt (noch) beherrschen zu können, ist die Einführung höherer Abstraktionsniveaus. Bei einem Komponentenmodell können durch Komposition mithilfe einer Komponentenhierarchie höhere Abstraktionsniveaus erreicht werden. Aber auch in Bereichen außerhalb der Softwareentwicklung findet diese Methode Anwendung, beispielsweise beim Design von Prozessoren [LDW04].

Um die steigende Komplexität von multifunktionalen Systemen beherrschen zu können, wird künftig die Modellierung der Systemfunktionalität im Sinne einer geeigneten Strukturierung der Funktionalität den Rahmen des Requirements Engineering erweitern [Bro03b]. Dies spiegelt sich auch beispielsweise bei modernen Vorgehensmodellen wie etwa dem V-Modell *xt* [VMX05, MRD+04, GDMR04] wider, bei dem Requirements Engineering Bausteine verstärkt an Bedeutung gewonnen haben. Voraussetzung für ein einheitliches Verständnis eines Systems ist idealerweise eine Anforderungsspezifikation,

welche vollständig die geforderten Systemeigenschaften, also die Anforderungen an das System, festlegt. Eine Anforderungsspezifikation sollte auch klar funktionale von nichtfunktionalen Eigenschaften trennen [Bro04d].

Gegenstand unserer Betrachtungen sind die funktionalen Anforderungen, die den Funktionsumfang eines Systems und damit das Systemverhalten aus Nutzungssicht bestimmen. Ähnlich wie Henning Genz [Gen04] die Herleitung physikalischer Gesetze aus einfachen Prinzipien darstellt,

> „... dass wir zahlreiche Aspekte des Modells durch Prinzipien verstehen, die selbst nicht mathematisch sind .... Insofern wir also das Modell durch nicht-mathematische Prinzipien verstehen, verstehen wir auch die Existenz und die Eigenschaften ...“,

sollte idealerweise auch das Wirkungsgefüge der Funktionen eines Softwaresystems mit einleuchtenden, intuitiven Mitteln beschrieben werden können. Damit ließen sich die Fragen, die sich ein kritischer Requirements Engineer stellen sollte, ob die aufgestellten Anforderungen zutreffend im Sinne der Nutzer sind, ob sie verständlich und im weiteren Prozess nutzbar sind, deutlich besser beantworten.

Durch eine strukturierte Modellierung der funktionalen Anforderungen soll dies nun erreicht werden. Dabei hilft eine strukturierte Sicht auf die Funktionen eines Systems, welche die Zusammenhänge dieser Funktionen offenbart. Neben der isolierten Modellierung der Nutzungsfunktionen mit den auch aus der UML [OMG04] bekannten Beschreibungstechniken wie Use Cases, Szenarien (Interaktionsdiagramme) und Zustandsmaschinen zur Beschreibung vollständigen Verhaltens werden insbesondere die Beziehungen und Abhängigkeiten zwischen den Funktionen explizit modelliert.

In Anlehnung an die in FOCUS [BS01] definierte Blackboxsicht beziehungsweise Glassboxsicht einer Komponentenarchitektur bezeichnen wir diese strukturierte Sicht als *strukturierte Blackboxsicht*. Dies soll zum Ausdruck bringen, dass wir bei der Strukturierung der Funktionalität zwar Einblick in das Zusammenspiel von Funktionen bekommen wollen. In diesem Zusammenhang interessiert jedoch nicht die Komponentenarchitektur, sondern die funktionalen Zusammenhänge verwendeter Funktionen, die zu einem bestimmten Blackboxverhalten führen.

1.1 Problembeschreibung

Der Sprung von den Anforderungen hin zum Entwurf eines Systems bleibt allerdings groß. Der direkte Übergang zu einem Systemdesign aus einer Menge von funktionalen Anforderungen erweist sich im Allgemeinen als sehr schwierig und fehleranfällig. Ideal wäre

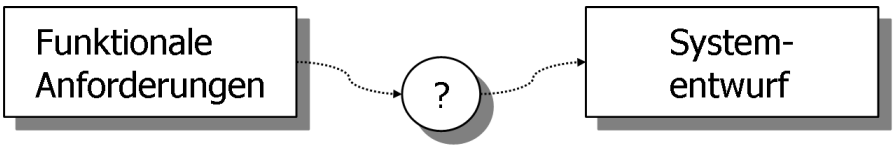


Abbildung 1.1: Lücke im Requirements Engineering?

selbstverständlich, wenn für diesen Übergang ein vollständig automatisiertes Verfahren zur Verfügung stehen würde. Durch den Einsatz von Modellen beim Requirements Engineering wird aber zumindest ein sanfterer Übergang zum Systemdesign ermöglicht. Für die typischen Fehlerquellen, wie das Übersehen von funktionalen Anforderungen und deren Zusammenhänge, gilt es Abhilfe zu schaffen. Mit der Anforderungsmodellierung wird der Bezug von Anforderungen zu späteren Systemmodellelementen frühzeitig hergestellt, welcher die systematische Verfolgbarkeit von Anforderungen erst ermöglicht. Die mit der Modellbildung erreichte systematische Erfassung der funktionalen Zusammenhänge trägt nicht nur dazu bei, das Wirkungsgefüge des Softwaresystems besser zu verstehen, sondern auch die Auswirkung von nachträglichen Änderungen der Anforderungen besser abschätzen zu können.

Was fehlt, ist ein integriertes Modell verschiedener Beziehungstypen, die sich aufgrund der wesentlichen Systemaspekte verteilter Systeme [Bro04c] und den sich daraus möglichen Sichten auf das System ergeben. Nur eine isolierte Sichtweise, wie beispielsweise die Kommunikation, reicht nicht aus, um die Beziehungen von Funktionen vollständig zu beschreiben. Es ist offensichtlich, dass eine Funktion f mit einer Funktion g in Beziehung steht, wenn f die von g erzeugten Ausgabenachrichten als Eingabe verarbeitet. Besteht aber auch eine Beziehung zwischen f und g, wenn die von g erzeugten Nachrichten von f nur weitergegeben werden [DGP+04a]?

Dienste sind (zunächst) voneinander isolierte, meist sogar unabhängig voneinander entwickelte Funktionalitäten. Abgesehen von einer Nutzungsbeziehung werden Dienstbeziehungen in der Regel nicht bewusst modelliert. Besonders deutlich wird dies am Beispiel von adaptiven Diensten, denen kein „Wissen“ über andere, konkurrierende Dienste zugrunde liegt. Durch eingeschränkte Sicht auf das Systemverhalten werden nur bestimmte System- oder Umgebungseigenschaften lokal optimiert. Dadurch dass mögliche Beeinflussungen des Verhaltens nicht bewusst modelliert sind, sind auch keine geeigneten Interaktionsschnittstellen vorgesehen. Betrachten wir dazu folgende Situation:

Beispiel (Transportdienst über Internet)
Eine angebotene Funktion der Transportschicht des OSI-Modells [Tan89] ist die fehlerfreie Übertragung. Dafür ist eine Funktion vorgesehen, die bei Verlust von Daten diese solange wieder sendet, bis alle Daten vollständig empfangen sind (Retransmit). Daneben

gibt es eine Funktion, welche die Überlastung des Übertragungsnetzes verhindern soll (Congestion Control) und gegebenenfalls das Senden von Daten verzögert.

Beiden Funktionen gemeinsam ist Optimierung der Übertragungsperformanz, allerdings aus unterschiedlichen Blickwinkeln. Während *Retransmit* den Durchsatz einer einzelnen Verbindung zu optimieren versucht, hat *Congestion Control* den Durchsatz des gesamten Netzwerks im Fokus. Unterstellt man beiden Funktionen die Unabhängigkeit, kann dies abhängig von der Übertragungsqualität der physikalischen Verbindung (etwa drahtgebundenes Netz oder drahtloses Netz) zu extremen Situationen führen: Ist die Übertragungsrate sehr schlecht, werden immer häufiger Daten wieder versendet (*Retransmit*), das zur Überlastung führen kann. Der Durchsatz wird nun unnötigerweise stark reduziert, weil sich die Verzögerungsintervalle für das Versenden von Daten stetig verlängert haben (*Congestion Control*).

Letzten Endes empfiehlt sich also doch keine Unabhängigkeitsbeziehung zwischen *Retransmit* und *Congestion Control* anzunehmen, sondern eine durch die Übertragungsqualität motivierte Beeinflussungsbeziehung zu fordern. □

Ein weiteres Beispiel ist bei eingebetteten Systemen, wie das Mobiltelefon, vorzufinden, bei denen auf der einen Seite hohe Performanz beziehungsweise Durchsatz im Vordergrund steht und andererseits nur vergleichsweise begrenzt Energieressourcen zur Verfügung stehen:

Beispiel (Leistung versus Auslastung beim mobilen Telefon)
Ziel der Steuerung der Leistungsaufnahme ist die höchste Akkulaufzeit. Bei Verringerung der Leistung werden Anwendungen deaktiviert. Dagegen ist es das Ziel der Steuerung der Netzauslastung, optimalen Netzdurchsatz durch Aktivierung von Anwendungen zu erreichen. Wie im Beispiel vorher stellt auch dies eine Konkurrenzsituation der Funktionen dar.

Auch dies kann zu einer Eskalation führen, wenn hier die latente Beziehung zwischen Leistungs- und Durchsatzsteuerung ignoriert (und nicht geeignet modelliert) wird. Das so mit sinkender Energiemenge beschleunigte Aufbrauchen der Energiereserven ist mit Sicherheit ein ungewolltes Verhalten. □

Beide Beispiele sind typisch für teilweise gegensätzliche Anforderungen. Derartige Konflikte entsprechen aber oft dem in Abbildung 1.2 illustrierten Beziehungsmuster. Die Funktionen f_1 und f_3 stehen dort etwa in einer Beziehung b_x und die Funktionen f_2 und f_3 in einer b_y - Beziehung.

- Sind die Beziehungen b_x und b_y zu ein und derselben Funktion f_3 überhaupt zulässig?
- Stehen dadurch auch f_1 und f_2 in einer bestimmten Beziehung?

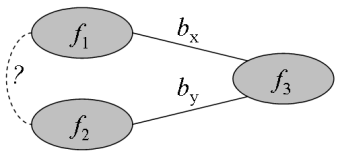


Abbildung 1.2: Gegensätzliche Anforderungen

- Kann durch eine explizit spezifizierte Beziehung zwischen f_1 und f_2, etwa durch die Definition einer geeigneten Interaktionsschnittstelle, eine der Beziehungen b_x oder b_y eliminiert werden?

Mit der Beantwortung dieser Fragen können mögliche Lösungen gefunden werden.

Davon abgesehen, dass gerade bei multifunktionalen Systemen das Design von Komponenten und der Komponentenarchitektur kaum direkt aus den Anforderungen heraus zu bewerkstelligen ist, haben so genannte „unwanted feature interactions“ auch meist ihre Ursache nicht in einer fehlerhaften Komponentenlogik oder Kommunikationslogik unter den Komponenten. Wie in den Beispielen gezeigt, führt dazu vielmehr die rein isolierte Betrachtung der geforderten Eigenschaften. Im Übrigen sind die Konsequenzen nicht erkannter Zusammenhänge für die Definition von Produktvarianten noch weitreichender.

Wenn wir ein Softwaresystem als eine Kombination einer Menge von funktionalen Systemeigenschaften betrachten, so entspricht im Falle der wechselseitigen Unabhängigkeit aller Funktionen das Softwaresystem trivialerweise stets einem „gültigen“ Produkt. Dies ist in der Regel allerdings nicht der Fall: Funktionen können andere Funktionen voraussetzen, Funktionen erst freischalten, Teilfunktion sein, das Verhalten anderer Funktionen beeinflussen, von anderen Funktionen kausal abhängig sein, oder beliebige weitere Beziehungen untereinander besitzen. Die gültigen Kombinationsmöglichkeiten werden dadurch eingeschränkt.

Es reicht daher nicht aus, geforderte Eigenschaften eines Systems nur durch funktionale und nichtfunktionale Anforderungen zu unterscheiden. Voraussetzung dafür ist aber erst die Festlegung geeigneter Sichtweisen auf ein Softwaresystem. Dazu gehört das Zugrundelegen eines geeigneten Produktmodells (siehe Abbildung 1.3), das insbesondere **aus Nutzungssicht** der Modellierung der Zusammenhänge der Systemeigenschaften Rechnung trägt.

Anhand dieses Produktmodells können Anforderungen nun bereits feiner strukturiert werden und durch Einteilung in die betroffenen Sichten klassifiziert werden. Die Nutzungssicht umfasst funktionale Anforderungen und das Systemverhalten aus Nutzungssicht (Nutzungsschnittstellen). Es ist damit allerdings nicht getan, die reinen Eingabe- und Ausgabemodalitäten festzulegen. Insbesondere sind dort Beziehungen und Abhän-

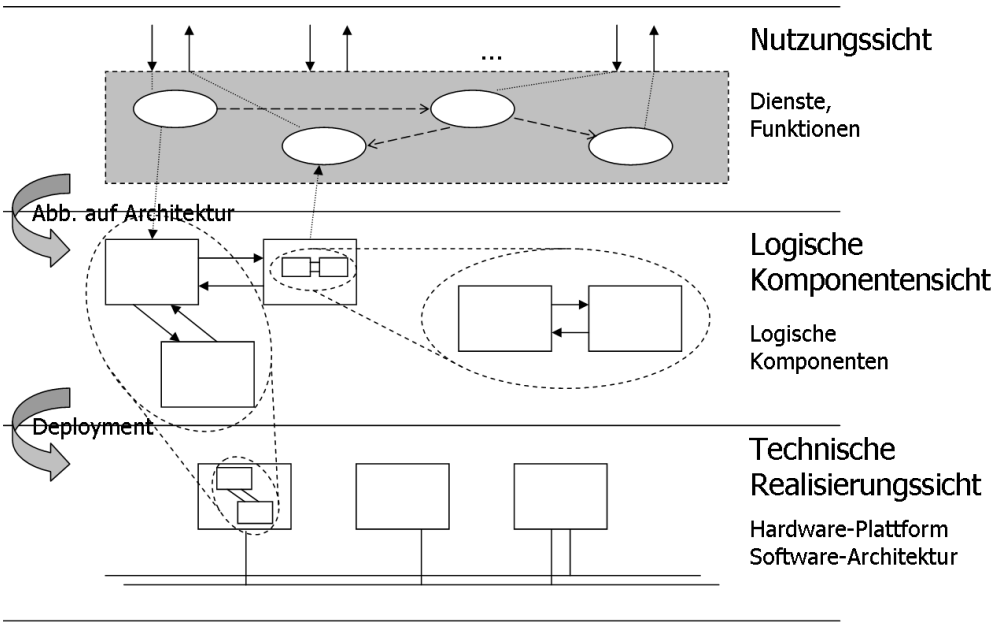


Abbildung 1.3: Produktmodell

gigkeiten zwischen den Funktionen, die sich auf das Zusammenspiel der Funktionen und damit auf das Systemverhalten auswirken, zu charakterisieren.

Die Komponentensicht erlaubt einen Blick auf die logische Architektur des Systems, also die Strukturierung in Teilsysteme und deren Verbindungen zum Austausch von Nachrichten (Komponentenschnittstellen). Mithilfe von Szenarien werden hier Interaktionen zwischen den logischen Komponenten beschrieben. Durch das Zusammenwirken der Teilsysteme wird die Gesamtfunktionalität des Softwaresystems erbracht.

In der technischen Realisierungssicht wird die Abbildung der logischen Komponentenarchitektur auf Softwarearchitekturen beschrieben und letztlich auch die Verteilung der Software auf entsprechende Elemente der Hardwareplattform.

Ein wichtiger Bestandteil des modellbasierten Requirements Engineering wird also eine Strukturierung der Systemeigenschaften auf der Nutzungsebene sein, welche die Festlegung der Zusammenhänge sowohl modellierungstechnisch als auch prozesstechnisch unterstützt. Diese Arbeit leistet dazu ihren Beitrag.

1.2 Begriffsdefinitionen

An dieser Stelle sollen nicht zu den aus der Softwaretechnik bekannten Begriffen noch weitere hinzugefügt werden oder bereits eingeführte Begriffe neu interpretiert werden. Allerdings werden einige Begriffe in der Literatur nicht eindeutig verwendet. Mit Vorsicht zu begegnen ist insbesondere dem Begriff „Dienst“ beziehungsweise „Service“. Dieser wird je nach Domäne und Anwendung unterschiedlich definiert [SS03b]. Genauso werden aber auch verschiedene Begriffe mehr oder weniger synonym für „Dienst“ verwendet [Bro03b]. Deshalb legen wir hier fest, wie die im Folgenden verwendeten Begriffe zu verstehen sind.

Anforderungen

Wie bereits oben erwähnt, sind funktionale von nichtfunktionalen Anforderungen zu unterscheiden. Gegenstand dieser Arbeit sind ausschließlich funktionale Anforderungen bei der Anforderungsanalyse.

Definition 1.1 (Funktionale Anforderung)

Eine funktionale Eigenschaft ist eine Eigenschaft eines Softwaresystems, die ein bestimmtes Teilverhalten beziehungsweise eine Teilfunktion des Systems charakterisiert.
Im Rahmen eines Pflichtenhefts gehört zur Beschreibung einer funktionalen Anforderung neben Ein- und Ausgabedaten und Schnittstellen zu anderen Systemen oder Benutzern das speziell in dieser Arbeit interessierende funktionale Verhalten. □

Mit „Feature“ wird in der Literatur nicht nur eine funktionale Eigenschaft, sondern oft auch die Umsetzung von nichtfunktionalen Anforderungen bezeichnet [PSC+01]. In dieser eingeschränkten Sichtweise[1] beschreiben nichtfunktionale Anforderungen zusätzliche, über die Funktionalität hinaus gehende Eigenschaften eines Softwaresystems, die das implizite Ergebnis einer Implementierung sind. Die Konsequenz einer nichtfunktionalen Anforderung kann zum Beispiel sein, dass nur eine bestimmte Abbildung von Funktionen auf Komponentenarchitekturen zulässig ist. Die Analyse derartiger Eigenschaften ist jedoch kein Bestandteil dieser Arbeit. Deshalb werden nichtfunktionale Systemeigenschaften hier nicht näher betrachtet.

Dienst, Nutzungsfunktion und Feature Interaction

Die Bedeutung des Begriffs „Dienst“ ist in vielen Verwendungen der Informatik unpräzise [Bro03a]. Allen Definitionen gemeinsam ist aber der Ausschluss von Struktur [SS03b]. Im Gegensatz zu komponentenorientierten Ansätzen sind mit einem Dienst keine strukturellen Eigenschaften verbunden. Das Verhalten ist also das entscheidende Kriterium.

[1] Nichtfunktionale Anforderungen können zum Beispiel auch Aspekte wie Prozesse, Werkzeuge und so weiter umfassen.

Da es uns hier auf die Spezifikation von Systemverhalten ankommt, verwenden wir im Rahmen dieser Arbeit die Begriffe „Funktion“ beziehungsweise „Nutzungsfunktion“ und „Dienst“ synonym und definieren in Anlehnung an [Bro03b,SS03a] einen Dienst informell:

Definition 1.2 (Dienst, engl. Service)

Ein Dienst ist eine Menge von Verhaltensmustern, welche die Nutzung eines Softwaresystems zu einem bestimmten Zweck charakterisiert.
Mit einem Dienst werden funktionale Systemeigenschaften aus Nutzungssicht beschrieben.
□

Dienste werden typischerweise durch Nutzungsfälle (engl. „Use Cases“) beschrieben. Wenn wir uns noch einmal Abbildung 1.3 in Erinnerung rufen, sind Dienste also Verhaltenseinheiten auf der Ebene der Nutzungssicht, mit denen sich das Systemverhalten strukturieren lässt. Durch die Abbildung auf logische Komponenten entsprechen Dienste in der Regel partiellem Schnittstellenverhalten von Komponenten. Präziser formuliert steht daher weniger ein Dienst im Blickpunkt des Interesses dieser Arbeit, als vielmehr die Funktionalitäten und deren Zusammenhänge im Sinne einer *Diensterbringung*. Eine Diensterbringung, also eine „von außen“ wahrnehmbare Funktion, kann durch Interaktionsmuster und die an der Interaktion beteiligten Rollen angegeben werden [KM04]. Auf diesem Abstraktionsniveau wird noch keine Komponentenarchitektur vorausgesetzt, bei der bereits ein Dienst von einer Komponente angeboten wird.

Eine Nutzungsfunktion ist geprägt durch die Interaktion, also durch den Austausch von Nachrichten, mit dem Funktionsnutzer. Es ist dabei zunächst unerheblich, welcher der Kommunikationspartner initiativ ist. Aus Nutzungssicht kann eine Funktion auch vom System aus initiiert werden, das heißt, die Initiative muss nicht notwendigerweise vom (menschlichen) Nutzer der Funktion ausgehen. Beispielsweise ist die Funktion «PIN eingeben» offensichtlich eine Nutzungsfunktion eines Mobiltelefons. Wie wir später sehen werden, wird sie allerdings niemals auf Initiative des menschlichen Nutzers hin genutzt werden, sondern wird stets nur vom System selbst ausgelöst.

Im Zusammenhang mit „Features“ fällt häufig auch der Begriff der „Feature Interaction“. Damit ist bei der Kombination von zwei oder mehreren funktionalen Systemeigenschaften eine ungewollte Beeinflussung gemeint. „Feature Interaction“ entspricht an sich dem Normalfall und bedeutet, dass Features nicht unabhängig voneinander sind und miteinander interagieren. Wenn diese Beeinflussung jedoch ungewollt, also „nicht spezifiziert“ stattfindet, bezeichnet man diesen Seiteneffekt auch genauer als so genannte „Unwanted Feature Interaction“ [Zav99a].

Definition 1.3 (Feature Interaction)

Die erwünschte oder unerwünschte Beeinflussung einer Funktion durch eine andere Funktion wird als „Feature Interaction“ bezeichnet.
Eine Funktion g habe ein spezifiziertes Verhalten $B(g)$ und in Kombination mit einer Funktion f ein Verhalten $B_f(g)$. g wird von f beeinflusst genau dann, wenn $B_f(g)$ von der Vorgabe in g abweicht: $B(g) \neq B_f(g)$. □

Sichten auf ein Softwaresystem

Sichten sind allgemein ein probates Mittel, nur speziell interessierende Information komplexer Sachverhalte hervorzuheben. Die mit einem Softwaresystem verbundenen Problemstellungen können so getrennt voneinander betrachtet werden. Mit unterschiedlichen Sichten wird ein Softwaresystem an und für sich strukturiert. Etablierte Prinzipien sind dabei etwa die Abstraktion, bei der unwesentliche Details unterdrückt werden, und die Projektion, bei der lediglich eine Teilmenge der Systemeigenschaften betrachtet wird [Par98].

Grundlegende Systemaspekte projizieren ein Softwaresystem etwa auf Verteilungseigenschaften oder Kommunikations- beziehungsweise Interaktionseigenschaften. Weitere Gesichtspunkte sind Koordination oder Zeit, die sich auf zeitliche oder kausale Abhängigkeiten konzentrieren. Einige wesentliche Sichten beschreiben dabei die Wechselwirkungen mit der Umgebung (Schnittstellen), Aktions- oder Ereignisfolgen (Ablauf), Datenstrukturen oder auch Zustände und Zustandsänderungen [Bro04c]. Wenn man an dem inneren Aufbau und an der inneren Funktionsweise eines Softwaresystems mit lokalen Zuständen und internen Aktionen interessiert ist, wird ein System durch die so genannte *Glassboxsicht* betrachtet. Bei dem in dieser Arbeit untersuchten Nutzungsverhalten steht allerdings die äußere Funktionsweise im Vordergrund, die durch eine Blackboxsicht erkennbar wird [BS01, Bro04c].

Definition 1.4 (Blackboxsicht)

Bei der Blackboxsicht wird die Wirkung eines Systems oder eines Teils eines Systems nach außen betrachtet, also das Ein-/Ausgabeverhalten als Reaktion auf Umgebungsereignisse. Das von außen beobachtbare Verhalten wird auch als Blackboxverhalten bezeichnet. □

Auch durch das in Abbildung 1.3 dargestellte Produktmodell wird ein System durch grundlegende Sichtweisen gegliedert. Nutzungsfunktionen und deren Abhängigkeiten (Nutzungssicht) abstrahieren von Komponentenarchitekturen (Komponentensicht) und diese wiederum von Softwarearchitekturen (Realisierungssicht). Wenn man den Architekturbegriff abstrakt als die Zergliederung eines Systems in Teilsysteme, sowie deren Zusammenspiel, auffasst, dann besitzen auch die betrachteten Konzepte der Nutzungssicht architekturellen Charakter. Wir sprechen deshalb im Folgenden auch von einer *Nutzungsarchitektur* oder von einer *logischen Dienstarchitektur*. Ein System nur aus der Blackboxsicht

heraus zu betrachten ist für die Nutzungsperspektive nicht ausreichend, da auch Aussagen über den Zusammenhang der Nutzungsfunktionen zu treffen sind.

Definition 1.5 (Nutzungsarchitektur)

Eine Nutzungsarchitektur strukturiert ein Softwaresystem in funktionale Systementitäten aus Nutzungssicht, deren Zusammenwirken über Beziehungen und Abhängigkeiten modelliert ist. □

Nutzungsfunktionen werden mithilfe von funktionalen Anforderungen formuliert. Um ein über das Blackboxverhalten hinaus gehendes Systemverständnis entwickeln zu können, ist dagegen zusätzliches Wissen um die funktionalen Zusammenhänge erforderlich. Die Glassboxsicht ist dafür jedoch nicht geeignet. Mit einer Nutzungsarchitektur wird also gewissermaßen die Blackboxsicht strukturiert.

Bei MMI-Systemen[2] ist typischerweise der Zugang zu den Funktionen des Systems über verschiedene Modalitäten möglich. Die Zusammenhänge der Funktionen werden dabei in der Regel umso komplexer, je mehr Zugänge zu den Funktionen, insbesondere auch zugleich mögliche Zugänge, angeboten werden. Eine Nutzungsarchitektur kann also auch ein hohes Maß an Komplexität aufweisen. Es ist deshalb sinnvoll, auch diese durch verschiedene Sichten zu strukturieren. Allerdings werden sich diese Sichten weniger auf technische Aspekte sondern mehr auf Nutzungsaspekte konzentrieren, die prinzipiell beliebig festgelegt werden können. Im weitesten Sinne wird ein Nutzungsaspekt einen bestimmten Kontext darstellen, der durch die Art und Weise der Nutzung (Anwendung, Management, Konfiguration), durch Zugänge zum Softwaresystem (Rollen, Bedienungsmöglichkeiten), durch strukturelle Vorgaben (Funktionsblöcke) oder durch andere nichtfunktionale Anforderungen ausgezeichnet ist.

1.3 Ansatz und Ergebnisse

Die Vision dieser Arbeit ist eine Methodik für den Entwurf von Softwaresystemen, deren Verhalten sich aus der konfliktfreien Kombination von Funktionen mit ausschließlich erwünschten „Feature Interactions“ ergibt. Das System kann im Sinne von Voraussagbarkeit beziehungsweise Nachvollziehbarkeit des Verhaltens einfach erweitert oder angepasst werden.

Die nicht erkannten Zusammenhänge und Abhängigkeiten von funktionalen Anforderungen sind in der Praxis ein häufig beobachtetes Problem. Die Problematik kommt noch deutlicher bei der Definition von Varianten eines Softwaresystems (Produktlinie) zum

[2] MMI ist die Abkürzung für „Mensch-Maschine-Schnittstelle“ (engl. Man Machine Interface).

Vorschein. Ohne explizites Wissen um die funktionalen Zusammenhänge können Produkte mit hinreichender Qualität, im Sinne geforderter funktionaler Eigenschaften, kaum gebildet werden. Nutzungsarchitekturen mit der expliziten Modellierung von Beziehungen anhand von Beziehungstypen sind bis dato ein offenes Forschungsthema (siehe auch Kapitel 2 und [Bro03a, Bro03b, Bro04a]).

Die meisten Modellierungstechniken sind hinsichtlich des Entwicklungsprozesses in der Entwurfsphase und bezüglich des Produktmodells in der logischen Komponentenarchitektur anzusiedeln. Ihre Zielsetzung sind ausführbare Modelle, um das Systemverhalten simulieren oder verifizieren zu können. Die explizite Formulierung von Verhaltensbeziehungen ist dort nicht vorgesehen und sie konzentrieren sich in der Regel auch nur auf architekturelle Aspekte. Wir wollen hier aber noch deutlich vor dem Design eines Softwaresystems eingreifen. Mit dieser Arbeit wird eine Methode zur Verfügung gestellt, die es erlaubt, die Anforderungsanalyse zu erweitern und bereits dort ein Beziehungsmodell zu entwickeln, welches den Übergang zum Design verbessert.

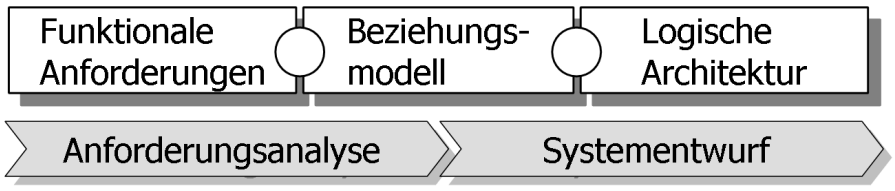


Abbildung 1.4: Modellierung von Funktionsbeziehungen als Übergang zum Systemdesign

In dieser Arbeit werden daher Konzepte vorgeschlagen, die dabei helfen, das Verhalten eines Softwaresystem aus Nutzungssicht nachvollziehen zu können. Ein Softwaresystem wird hier durch Funktionen sowie explizit definierte Funktionsbeziehungen strukturiert.

Auf der Grundlage von Untersuchungen bestehender Modellierungstechniken und von theoretischen Überlegungen wird ein Satz nützlicher Beziehungstypen präsentiert und formal fundiert. Ein Beziehungstyp umfasst dabei die wesentlichen Eigenschaften eines Zusammenhangs zwischen kombinierten Funktionen. Eine Funktion ist entweder unabhängig von einer anderen (independent), wird von einer anderen beeinflusst (affect) oder ist Teilfunktion einer anderen (subfunction). Neben der Kombinierbarkeit von Funktionen an sich sind dies die drei grundlegenden Beziehungstypen.

Um den Nutzen der Arbeit und die praktische Anwendbarkeit der Konzepte zu demonstrieren, werden im Rahmen einer Fallstudie Zusammenhänge zwischen Funktionen analysiert und entsprechend als Funktionsbeziehungen formuliert.

1.4 Inhalt und Aufbau

In diesem Abschnitt wird ein kurzer Abriss des Inhalts und des logischen Aufbaus der Arbeit gegeben.

Verwandte Arbeiten
: Im Anschluss erfolgt eine ausführliche Betrachtung verschiedener Arbeiten und Techniken vor allem aus den Bereichen des Requirements Engineerings und der Beschreibung von Architekturen. Hier wird eine Reihe von Ansätzen untersucht, um einerseits Ideen aus anderen Arbeiten aufzugreifen, und andererseits möglichen, auch implizit definierten, immer wiederkehrenden Beziehungstypen auf die Spur zu kommen. Die Betrachtungen umfassen eine Auswahl etablierter Beschreibungstechniken des Requirements Engineerings, sowie neue, modellbasierte Ansätze des Requirements Engineerings. Darüber hinaus werden verschiedene Architekturen und Architekturbeschreibungen untersucht, genauso wie aktuelle dienstbasierte Modellierungstechniken. Auch solche Arten von Beziehungen, die im Zusammenhang mit Produktvarianten oder Plattformen zu sehen sind, werden diskutiert.

Grundlagen
: In diesem Kapitel werden die verwendeten Notationen vorgestellt. Einige formale Grundlagen bilden den Ausgangspunkt für die Entwicklung der Konzepte einer Nutzungsarchitektur. Auf der Basis von Nachrichtenströmen, wie sie aus der FOCUS Theorie [BS01] bekannt sind, werden hier insbesondere die Begriffe der Kanalhistorien, des Schnittstellenverhaltens und der Verhaltensverfeinerung eingeführt. Da Nutzungsfunktionen meist partiell definiert sind, taucht hier speziell der Begriff der Funktionsdomäne auf. Schließlich wird zwischen einem Beziehungstyp und einer konkreten Funktionsbeziehung unterschieden.

Grundlegende Funktionsbeziehungen
: Um zu einer Nutzungsarchitektur zu gelangen, ist die Blackboxsicht auf ein System differenzierter zu betrachten. Nutzungsfunktionen und die Beziehungen zwischen diesen Funktionen strukturieren eine Blackboxsicht. Verschiedene Nutzungsaspekte werden dazu verwendet, Funktionsbeziehungen zu klassifizieren. Funktionen können hierarchisch strukturiert sein oder sie sind schichtenartig im Sinne einer Nutzung geordnet. Aus Verhaltenssicht schließlich können Funktionen andere Funktionen beeinflussen.

  Für Subfunktions- beziehungsweise Superfunktionszusammenhänge, wie sie auch aus Funktionsnetzen bekannt sind, wird hier der entsprechende Beziehungstyp definiert. Der zentrale Begriff der Kombinierbarkeit stützt sich ebenfalls auf diesen Beziehungstyp und wird als Eigenschaft eines Supersystems formuliert. Eine ganzheitliche Betrachtung, bei der grundsätzlich ein Supersystem mit einbezogen wird,

ist die Voraussetzung zur Identifizierung von Funktionsbeziehungen. Die Domäne eines Supersystems wird gemäß des Verhaltens einer zu betrachtenden Funktion partitioniert und zur Beschreibung von Beziehungstypen verwendet. Zum Abschluss des Kapitels wird die Unabhängigkeit von Funktionen ebenfalls als eine grundlegende Beziehung zwischen Funktionen diskutiert.

Das Verhalten kombinierter Funktionen
: In diesem Kapitel werden Zusammenhänge zwischen Funktionen diskutiert, welche die verschiedenen Arten des Einflusses auf das Verhalten charakterisieren. Bei diesen Beziehungstypen wird grundsätzlich die Kombinierbarkeit der in Beziehung stehenden Funktionen vorausgesetzt. Aber auch für an sich nicht kombinierbare Funktionen, wenn es also widersprüchliche Ausgabeanforderungen gibt, werden Möglichkeiten gezeigt, wie sie dennoch unter bestimmten Voraussetzungen kombiniert werden können. Unter dem Begriff der *bedingten Kombinierbarkeit* werden ebenso Beziehungen im Kontext der Zugangskontrolle vorgestellt. Schließlich werden Zusammenhänge zwischen Funktionen untersucht, die von den Nutzern der Funktionen zwar als nicht unabhängig wahrgenommen werden, jedoch eher speziellen Nutzungsmustern gleichkommen.

Fallstudie
: Ausgewählte Nutzerfunktionen eines Mobiltelefons bilden den Rahmen der Fallstudie. Für diese konkreten funktionalen Anforderungen wird eine Nutzungsarchitektur entwickelt und entsprechende Spezifikationen als Zustandsautomaten mit Ein- und Ausgabe gezeigt. Alle vorher definierten Beziehungstypen werden dabei anhand von Beispielen ausführlich diskutiert und zur Beschreibung der Zusammenhänge der Nutzungsfunktionen verwendet.

Zusammenfassung und Ausblick
: Im abschließenden Kapitel werden die Ergebnisse der Arbeit zusammengefasst und unter anderem dem Ansatz der „Distributed Feature Composition“ (DFC) [JZ03] gegenüber gestellt. Darüber hinaus wird ein Ausblick auf weitere interessante Fragestellungen gegeben, wie zum Beispiel die Entwicklungsmethodik einer Nutzungsarchitektur, die Beschreibung komplexerer Zusammenhänge oder die Bereitstellung eines Kalküls von Beziehungen.

Kapitel 2

Verwandte Arbeiten

Die Identifizierung von Beziehungen zwischen den funktionalen Eigenschaften eines Systems setzt zunächst eine geeignete Modellierung der funktionalen Anforderungen voraus. Zwar gibt es für die Beschreibung des Verhaltens eines Systems oder von Systemfunktionalitäten eine Vielzahl von Konzepten. Im Rahmen einer Anforderungsanalyse sind diese allerdings zur Beschreibung von funktionalen Zusammenhängen nicht ausreichend.

Sobald Anforderungen nicht mehr nur in Form von Prosa vorliegen, sondern als Elemente eines Modells, kann die Frage nach den Zusammenhängen zwischen den funktionalen Anforderungen beziehungsweise zwischen den Funktionen logisch angegangen werden. Anforderungen bereits während der Analysephase zu modellieren, ist ein vergleichsweise neuer Ansatz. Es gibt daher aktuell nur sehr wenige Arbeiten, die sich explizit mit dieser Thematik auseinander setzen. Insbesondere sind dafür geeignete Modellierungsumgebungen erst am Entstehen.

Außer ersten Ansätzen, das Requirements Engineering modellbasiert anzugehen, wird keine explizite Modellierung von funktionalen Zusammenhängen aus der Nutzungsperspektive unterstützt. Das Gros der Ansätze richtet den Blick auf (logische) Komponentenarchitekturen und ist mehr dem Entwurf von Softwaresystemen zuzuordnen. Es ist daher nicht überraschend, dass die Gemeinsamkeiten der meisten Ansätze auf der Dekomposition eines Systems und der Kommunikation zwischen den Systemteilen liegen.

Trotzdem untersuchen wir im Folgenden unterschiedliche Ansätze für verschiedene Problemstellungen, um – wenn auch implizit gegebene – mögliche, immer wieder kehrende, Beziehungen zu identifizieren, die als explizit modellierbare Beziehungstypen zur Verfügung gestellt werden sollten. Falls geeignet, werden dabei traditionelle Strukturierungen von Systemeigenschaften als Basis für die Weiterentwicklung wieder verwendet. Wir betrachten zunächst eine Auswahl etablierter Beschreibungstechniken des Requirements

Engineerings, sowie neue, modellbasierte Ansätze des Requirements Engineerings. Typische Beziehungen, wie sie bei verschiedenen Architekturen vorkommen, werden im Anschluss diskutiert. Weiterhin wird untersucht, welchen Beitrag aktuelle dienstbasierte Modellierungstechniken leisten. In deren Zusammenhang taucht oft der Begriff der „Feature Interaction“ auf. Welche Beziehungstypen im Zusammenhang mit Konfigurationen und Produktvarianten eine Rolle spielen werden schließlich ebenso beleuchtet, wie die Eigenschaften von Middlewareplattformen und Betriebssystemen.

Zum Abschluss werden in der Zusammenfassung die identifizierten Beziehungstypen nochmals überblicksartig erfasst. Auch noch zu ergänzende Beziehungstypen werden diskutiert, die dann im Hauptteil der Arbeit Anwendung finden.

2.1 Konzepte des Requirements Engineerings

Das Repertoire an Formalismen und Konzepten im Requirements Engineering zur Beschreibung interagierender Systeme ist umfangreich [Par98]. Je nach Betrachtungsweise (Systemaspekt [Bro04c]) des zu modellierenden Systems ist ein Konzept besser geeignet als das andere. Aus den darstellbaren Systemaspekten, wie Komponentenstrukturen, funktionales Verhalten, Abläufe, Steuerung und Zeit, greifen wir verschiedene Repräsentanten heraus, und untersuchen die jeweiligen modellierbaren Beziehungen, die auch für die Zusammenhänge aus Nutzungssicht von Interesse sind. Nicht formale Methoden, bei denen Anforderungen nur textuell erfasst und verwaltet werden können – wie es etwa das Werkzeug DOORS der Firma Telelogic [DOO05] bietet –, werden außen vor gelassen. Die untersuchten Konzepte sind allesamt formale Methoden. Einzige Ausnahme einer semiformalen Methode ist die Unified Modeling Language (UML) [OMG04] als prominenter Vertreter objektorientierter Beschreibungstechniken.

Komponentenstrukturen

Eine wesentliche Systemsicht richtet sich auf die Struktur eines Softwaresystems. Die Dekomposition eines Systems beschreibt die Systemteile und deren hierarchische Struktur. Klassische Modelle sind zum Beispiel Komponentenarchitekturen, bei denen die Darstellung von Komponenten und deren Zusammenhänge im Vordergrund stehen. Typische Darstellungsformen dafür sind Dekompositionsdiagramme oder Jacksondiagramme [Par98].

Die meist implizit gegebenen Beziehungen werden grafisch meist durch eine entsprechende Anordnung abgebildet. Eine implizite Beziehung „besteht aus“ impliziert eine streng hierarchische Struktur. Jacksondiagrammen erlauben darüber hinaus die Anwendung der Kompositionsprimitive *Sequenz*, *Auswahl* und *Wiederholung*. Damit wird die Interpretation der „besteht aus“ Beziehung entsprechend erweitert.

Funktionale Eigenschaften lassen sich ebenfalls durch Aufteilung beziehungsweise Zusammensetzung strukturieren. Die im Zusammenhang mit dieser Arbeit interessante Beziehung ist die *Teilfunktions*beziehung, die allerdings eher den Beziehungen hinsichtlich funktionalen Verhaltens unter dem Stichwort „Funktionsbaum“ zuzuordnen ist.

Funktionales Verhalten, Abläufe

Ein Funktionsbaum besteht aus Bezeichnungen von funktionalen Einheiten, die über verschiedene Beziehungen miteinander verbunden sind. Mit den implizit gegebenen Beziehungen werden streng hierarchische Zusammensetzungen, also funktionale Zerlegungen, beschrieben. Typische Beziehungen sind etwa Aufruf- oder Enthält-Beziehungen. Strukturelle Zusammenhänge werden damit nicht modelliert.

Dafür eignen sich die bereits erwähnten Jacksondiagramme, mit denen strukturelle und ablaufbezogene Zusammenhänge zugleich darstellbar sind. Allerdings sind für die Modellierung von Ablaufverhalten dann die Beziehungen entsprechend anders zu interpretieren, etwa als „führt aus“. Sequenzielle, an Bedingungen geknüpfte Abläufe innerhalb von Komponenten können auch mit Ablaufdiagrammen dargestellt werden. Aus der Nutzungsperspektive spielen Komponenten und auch deren innere Abläufe keine Rolle. Die Struktur des Blackboxverhaltens des Softwaresystems und damit eine mögliche Ordnung von Funktionen, die letztendlich von Komponenten angeboten werden, verdient dagegen Beachtung.

Die Kommunikation zwischen Entitäten oder, allgemeiner formuliert, das Zusammenspiel von verteilten Entitäten eines Systems kann durch Sequenzdiagramme oder „Message Sequence Charts“ (MSC [ITU99]) modelliert werden. Vertikale Linien stellen beispielsweise asynchron laufende Prozesse oder logische Komponenten eines logisch oder physikalisch verteilten Systems dar. Pfeile kennzeichnen ausgetauschte Nachrichten zwischen den Entitäten, wobei die Reihenfolge der Pfeile mögliche Interaktionssequenzen zwischen den Entitäten definiert.

Daneben werden so genannte „Hierarchical Message Sequence Charts“ [HPR97] oder auch „High-Level Message Sequence Charts“ [Krü04] (HMSC) verwendet, um durch Hintereinanderschaltung und Abstraktion beziehungsweise Detaillierung MSCs zu höherwertigen Funktionen (komplexeren Szenarien) zu komponieren. Knoten in einer HMSC sind Referenzen zu anderen (H)MSCs. Mit einer HMSC kann damit ein Use Case formuliert werden, der mehrere Szenarien beinhaltet.

Ein Szenario ist eine mögliche Komposition von MSCs, also ein mögliches Teilverhalten eines Use Case. Zwar ist eine referenzierte (H)MSCs prinzipiell auch Teil des Use Case, allerdings eher im Sinne einer Verhaltensdetaillierung denn eines Teilverhaltens. Analog den „mechanisms“ der IDEF0 (siehe unten), welche die gemeinsame Nutzung von spezifischeren Funktionen erlauben, können so Verhaltensspezifikationen wieder verwendet werden.

Da mit HMSCs die Kompositionsmöglichkeiten immer noch vergleichsweise elementar sind, werden in [KM04] die HMSCs um Kompositionsoperatoren erweitert, mit denen Interaktionen durch alternative oder wiederholte Interaktionsmuster komponiert werden können. Für die Diskussion dieser und anderer Erweiterungen von (H)MSCs im Zuge der Standardisierung der UML 2.0 sei hier auf den Abschnitt „UML Interaktionen“ (Seite 21) verwiesen.

Das mit Interaktionsmustern definierte Verhalten wird in [KM04] als „Dienst“ bezeichnet. Sowohl Nachrichten als auch Systementitäten können dort parametrisiert sein, die erst durch geeignete Instanzen von Interaktionsmustern ersetzt werden. Die an einer Interaktion beteiligten Systementitäten werden zunächst nur als so genannte „roles“ identifiziert, die erst bei der Erstellung einer Architektur auf Komponenten abgebildet werden.

Der konkrete Ablauf eines Diensts, sei er alternativ oder wiederholt, ist bei unseren Betrachtungen kein vorrangiges Kriterium. Entscheidend ist hier die Tatsache, dass Dienste komponiert oder die Zusammenhänge mit einer Vorgängerbeziehung beschrieben werden. Mit der Einführung von Rollen statt konkreter Komponenten als Kommunikationspartner ist die Notwendigkeit gegeben, dafür eine kompakte Spezifikationsform anzubieten. Dies wird durch die Bereitstellung einer generischen Notation (mit Wildcards) erreicht. Die dadurch erreichte Flexibilität beim Deployment von Diensten auf Komponentenarchitekturen ist in dieser Arbeit nicht im Fokus des Interesses. Deshalb wird dies nicht berücksichtigt.

Aus Nutzungssicht ist die (gemeinsame) Nutzung von Funktionen durch (höherwertige) Funktionen ein ebenso zu berücksichtigender Aspekt wie die (hierarchische) Strukturierung von Funktionalitäten durch eine *Teilfunktion*sbeziehung. Die Teilfunktionsbeziehung ist ein grundlegender Beziehungstyp, der auch typisch für Funktionsbäume ist.

Steuerung

Das Verhalten einer Funktion oder eines Systems kann ferner durch Zustände und Zustandsänderungen charakterisiert werden. Die Änderung eines Zustands wird durch das Eintreten bestimmter Ereignisse ausgelöst und ist in der Regel mit entsprechenden Aktionen verbunden. Durch eine Menge formulierter Bedingungen kann der gesamte Systemzustandsraum partitioniert werden. Diese Partitionen entsprechen den für das Auslösen von Aktionen interessanten Zustandsbeschreibungen. Entscheidungsbäume oder Zustandsautomaten sind hierfür geeignete Notationen. Durch eine Folge von Zuständen wird so das Verhalten sequenzieller Systeme beschreiben.

Wenn ein Zustand mehrere Folgezustände nach sich ziehen kann, werden damit Nebenläufigkeiten beschrieben, wie sie beispielweise bei Petrinetzen [Bau90] zur Anwendung kommen. Kausale Abhängigkeiten von Aktionen werden dort über die Synchronisierung von Stellen und Marken modelliert. Varianten von Petrinetzen erlauben die hierarchische Strukturierung durch Unternetze für Stellen und Transitionen (Abstraktion von

Verhaltensdetails) oder die Verwendung komplexer spezifizierbarer Marken (Datentypdefinitionen), wie es etwa bei Coloured Petri Nets [Jen98, KCJ98] vorgesehen ist.

Schließlich ist auch möglich, Verhalten durch Folgen von Ereignissen auszudrücken. Die Semantik der „Communicating Sequential Processes“ (CSP) [Hoa85] stützt sich darauf ab. Ein Prozess beschreibt dort allgemein das Verhalten einer Systementität. Abgesehen davon, dass Prozesse im Grunde durch Ereignisse gesteuert werden, können Prozesse sequenziell, parallel oder alternativ komponiert werden. Darüber hinaus sind noch Mechanismen zur nichtdeterministischen Auswahl von Prozessen vorgesehen.

Die Bedingungen, ob eine Funktion zur Verfügung steht oder (im Sinne einer ausgelösten Aktion) zur Ausführung kommt, kann durch kausale Abhängigkeiten abstrahiert werden. Die Detaillierung von Verhaltensbeschreibungen entspricht einer Nutzungsbeziehung.

Auch wird – gerade im Zusammenhang mit multifunktionalen Softwaresystemen – die konsistente Beschreibung von Zuständen immer schwieriger je größer die Komplexität der Datenmodelle ist. Inkonsistente Daten(zustände) sind besonders kritisch bei Funktionen, die nicht unabhängig voneinander sind.

Zeit

Zeitliche Anforderungen werden üblicherweise nicht isoliert beschrieben, sondern treten stets in Kombination mit anderen Systemaspekten auf. Daher sind temporale Eigenschaften als Erweiterung der oben beschriebenen Techniken zu sehen. Beispielsweise können mit Interaktionsdiagrammen unter Hinzunahme der Zeit die Dauer von Operationen oder die zeitlichen Zusammenhänge von ausgetauschten Nachrichten angegeben werden. Zeitliche und kausale Zusammenhänge zwischen nebenläufigen Prozessen können etwa durch *Ablaufdiagramme mit Zeit* [Par98] dargestellt werden.

Auch temporallogische Beschreibungen dienen der Formulierung zeitbezogener Aussagen und Prädikate. Dazu stehen eine Reihe von Operatoren zur Verfügung, welche relative Aussagen bezüglich zeitlicher Referenzpunkte erlauben. Wenn wir die Anwendung einer Funktion als Ereignis interpretieren, kann demnach $\mathsf{always}\ (f)$ verstanden werden als „f ist zu allen Zeitpunkten nutzbar“. Eine kausale Beziehung zweier Funktionen f und g implizierte eine zeitliche Abhängigkeit derart, dass etwa f beendet sein muss, bevor g genutzt werden kann: $\mathsf{always}\ (f \rightarrow \mathsf{sometimes}\ g)$.

Um die Modellierung für den praktischen Anwender zu vereinfachen wurde in [DAC98, DAC99] eine Reihe von Spezifikationen ausgewertet, um daraus typische Muster zu extrahieren und zur Verfügung zu stellen. Dies führte im Wesentlichen zu zwei Arten von Mustern, um zeitliche Abhängigkeiten zwischen Zuständen oder Ereignissen zu modellieren: so genannte „occurrence patterns“ und „order patterns“. Unterschieden werden dabei noch zeitliche Geltungsbereiche („scope“) wie die globale Gültigkeit, die Gültigkeit vor und nach Eintritt eines Ereignisses oder die Gültigkeit zwischen zwei Ereignissen. Aus-

sagen darüber, wie zwei verschiedene Zustände zeitlich zueinander in Beziehung stehen, lassen sich mit den Reihenfolgemustern „precedence“ und „response“ ausdrücken.

Der zeitliche Zusammenhang von funktionalen Eigenschaften aus der Nutzungsperspektive ist also insofern von Interesse, wenn sich das kombinierte Verhalten aus einer kausalen Abhängigkeit zwischen Funktionen ergibt. Relevant für Beziehungen zwischen Funktionen ist hier, dass eine Funktion die Voraussetzung für eine andere Funktion ist. Dies kann dahingehend ausprägt sein, dass eine Funktion erst die Voraussetzung für eine andere Funktion schafft oder eine Funktion einer anderen folgt.

Integrierte Ansätze

Neben den oben beschriebenen Ansätzen, die einzelne Sichtweisen auf ein System erlauben, gibt es eine Reihe von integrierten Ansätzen, welche mehrere Systemaspekte kombinieren (zum Beispiel Systemstruktur und funktionales Verhalten). Die UML und AutoFocus mit Focus als formaler Grundlage bieten solch einen integrierten Ansatz.

Unified Modeling Language (UML)

Die UML [OMG04] ist die Kombination einer Vielzahl verschiedenartiger, objektorientierter Konzepte. Der Sprachkern beim Einsatz in der Praxis allerdings besteht im Wesentlichen aus vier Beschreibungstechniken [RS01]: die Strukturdarstellung, die Darstellung von Anwendungsfällen, Verhaltensbeschreibungen und die Darstellung von Interaktionen mit den jeweils einschlägigen Diagrammarten. Im Rahmen der Analyse funktionaler Anforderungen erscheint insbesondere die Modellierung von Anwendungsfällen und von Interaktionen am passendsten, um weitere mögliche, nützliche Beziehungstypen zu entdecken.

UML Anwendungsfälle

Die UML bietet zur Beschreibung von Beziehungen zwischen Nutzungsfällen die beiden Konzepte `«include»` und `«extend»` innerhalb von Use Case Diagrammen an. An diesen Beziehungstypen sieht man deutlich, dass die Relationen im Allgemeinen nicht symmetrisch sind.

`«include»` wird in der Regel zur Redundanzvermeidung eingesetzt, um gleiche Teilfunktionalitäten in anderen Nutzungsfällen zu nutzen. Das Verhalten eines Nutzungsfalls ändert sich durch das Verhalten des (an einer definierten Stelle) eingefügten Nutzungsfalls. Wenn mit f_1 das Verhalten eines Nutzungsfalls und mit f_2 das Verhalten des einzufügenden Nutzungsfalls bezeichnet wird, dann ist f_1 abhängig von f_2, da f_2 von f_1 benötigt wird. Prinzipiell kann so auch eine hierarchisch funktionale Zerlegung gebildet werden. Zu beachten ist hier allerdings, dass mit `«include»` ein anderer Beziehungstyp als die Teilfunktionsbeziehung dargestellt wird. f_2 wird zwar von f_1 benötigt, aber f_2 muss deshalb noch keine von f_1 angebotene Teilfunktion sein.

Mit «extend» kann dagegen optionales Verhalten ausgedrückt werden, welches an bestimmte Bedingungen geknüpft ist. Auch hier wird das Verhalten eines Nutzungsfalls durch einen anderen geändert. Wenn mit f_1 wieder das Verhalten eines Nutzungsfalls und mit f_2 das Verhalten des erweiternden Nutzungsfalls bezeichnet wird, besteht hier allerdings die umgekehrte Abhängigkeit: f_2 ist abhängig von f_1, da sich f_2 nur dann auswirkt, wenn f_1 existiert.

Mit beiden Konzepten kann im Wesentlichen eine unbedingte («include») beziehungsweise bedingte («extend») Verhaltensänderung durch Eingriff in den Ablauf von Teilfunktionalitäten modelliert werden.

Wie sich aus Untersuchungen verschiedener UML Modelle kommerzieller Produkte ergeben hat [Ber04a], führt der Einsatz dieser Konzepte ohne Einhaltung bestimmter Anwendungsregeln nicht zwangsläufig zu semantisch korrekten Modellen. In [Ber04a] werden daher Techniken zur Analyse von UML Modellen diskutiert, bei denen unter anderem zwischen so genannten „leaf use cases“, „concrete use cases“ und „abstract use cases“ unterschieden wird. Letztere sind ausschließlich mit «include» und «extend» zusammengesetzte Nutzungsfälle. Konkrete Nutzungsfälle sind darüber hinaus durch Sequenzdiagramme definiert. Die *leaf use cases* fügen weder andere Nutzungsfälle ein noch werden sie erweitert und sind durch Zustands- oder Aktivitätsdiagramme festgelegt.
Die Idee von abstrakten und atomaren Funktionen findet sich auch im Zusammenhang mit der Kombination und der Dekombination von Funktionen wieder.

UML Interaktionen

Der Austausch von Nachrichten zwischen Systeminstanzen kann in der UML 2.0 durch UML Interaktionen, ein Dialekt der High-Level Messages Sequence Charts, beschrieben werden. Für Interaktionsdiagramme wie etwa *Sequence Diagrams*, *Interaction Overview Diagrams* oder *Communication Diagrams* bietet dafür die UML 2.0 die Möglichkeit, so genannte „Interaktionsfragmente“ zu kombinieren. Die wesentlichen Operatoren sind dabei alt, seq und loop, die zur Modellierung von alternativen, sequenziellen und wiederholten Abläufen dienen, wie sie unter anderem aus Jacksondiagrammen (siehe Seite 16) bekannt sind.
Betrachten wir nun diese Interaktionsfragmente hypothetisch als isolierte Funktionalitäten, die potenziell miteinander in Beziehung stehen können. Anhand der von der UML 2.0 angebotenen Kombinationsoperatoren untersuchen wir, ob sich daraus weitere Ausprägungen grundlegender Beziehungstypen gewinnen lassen.

Aus Alternativen (alt) kann stets nur ein Verhalten ausgewählt werden. Spezielle Alternativen sind optionales Verhalten und unterbrochenes Verhalten, die als abkürzende Schreibweisen (opt, break) angeboten werden. Optionales Verhalten kann interpretiert werden als zwei Alternativen, wobei eine Alternative dem leeren Verhalten entspricht. Eine Unterbrechung entspricht ebenfalls zwei Alternativen, bei der eine Alternative jedoch aus dem ursprünglichen Verhalten und die andere Alternative aus einem neuen

Verhalten besteht.
Alternative Abläufe kommen nur bei bestimmten, erfüllten Bedingungen („guard expressions“) zur Ausführung. Übertragen auf den Zusammenhang von Funktionen aus Nutzungssicht folgt eine Funktion einer anderen, wobei die Erfüllung einer Bedingung auch als die Schaffung von Voraussetzung zur Nutzung der Folgefunktion aufgefasst werden kann. Optionales Verhalten erweitert gegebenenfalls die Funktionalität. Durch Unterbrechung steht aus der Perspektive der Nutzungssicht eine bestimmte Funktionalität nicht mehr zur Verfügung.

Bei parallel (par) zusammengefügten Funktionen stellt das kombinierte Verhalten ein beliebig verschränktes oder überlappendes Verhalten der einzelnen Funktionen dar. Die Funktionen sind also wechselseitig unabhängig voneinander und stören sich nicht.

Bei sequenziell verbundenen Funktionen folgt das Verhalten einer Funktion auf das der anderen. UML 2.0 trennt hier noch zwischen so genanntem „weak sequencing“ (seq) und „strict sequencing“ (strict). Diese Feinheit der Unterscheidung spielt allerdings tatsächlich erst eine Rolle bei der Modellierung von Nachrichten- beziehungsweise Ereignissequenzen. Die schwächere Definition der Sequenz erlaubt prinzipiell – wie bei der Parallelkombination – die Überlappung von Verhalten. Allerdings ist (auf dem Niveau einzelner Nachrichten) gefordert, dass bezüglich gleicher Ereignisse pro involvierter Entität („lifeline“) die Ereignisse der Vorgängerfunktion vor denen der Nachfolgerfunktion erfolgen. Wenn die Ereignismenge disjunkt ist, entspricht seq dem par Operator. Bei einer strikten Sequenz gibt es keine Überlappung von Nachrichten oder Ereignissen.
Die Notwendigkeit einer Festlegung (im Sinne einer Anforderung an das Ablaufverhalten) von teilweise überlappendem Verhalten von Funktionen ist aus der Nutzungsperspektive zunächst schwer erkennbar. Diese spezielle Ausprägung des „weak sequencing“ kann unserer Erfahrung nach erst ab einem gewissen Detaillierungsgrad des Systems zur Anwendung kommen. Aus Nutzungssicht relevant ist deshalb an dieser Stelle der strict Operator. Abstrahiert von einem speziellen Ablauf ist hier von Interesse, dass das kombinierte Verhalten von Funktionen einer bestimmten Ordnung der Teilverhalten unterworfen ist. Dabei ist noch nichts darüber ausgesagt, wie diese Ordnung definiert ist: kausal, zeitlich, oder auch realzeitlich.

Wenn ein Verhalten aus der Sequenz ein und derselben Funktion definiert werden soll, kann dies abkürzend durch den (loop) Operator ausgedrückt werden. Die damit im Wesentlichen rekursive Anwendung des Sequenzoperators spielt für Beziehungen zwischen Funktionen aus Nutzungssicht nur eine untergeordnete Rolle.

Die mit (neg) markierten Interaktionsfragmente kennzeichnen spezielle Fälle von nicht gewünschtem Verhalten. Übertragen auf funktionelle Anforderungen aus Nutzungssicht ist dies eine Beschreibung von kritischen Kombinationen von Funktionen, also letztlich von unerwünschten und damit von zu vermeidenden Funktionszusammenhängen. Allerdings bietet die Semantik dieses Operators Spielraum für etliche Interpretationen [CK04].

Die Idee negativ formulierter Beziehungen zwischen Funktionen erscheint insbesondere vor dem Hintergrund der Entwicklung eines Beziehungskalküls interessant, um etwa widersprüchliche abgeleitete Beziehungen zu identifizieren.

Schließlich gibt es noch einen Satz von Operatoren, welche die Kombinierbarkeit von Interaktionsfragmenten einschränken. Der Vollständigkeit halber seien sie hier aufgeführt: ein kritischer Bereich (region) lässt keine Überlappung zu, Nachrichten können explizit eingeblendet (consider) oder ausgeblendet (ignore) werden und ein bestimmtes Folgeverhalten kann zugesichert (assert) werden. Diese Operatoren setzen genauso wie (weak) einen Detaillierungsgrad voraus, aus denen sich keine neuen Erkenntnisse für Beziehungstypen aus Nutzungssicht ergeben. Sie sind hier deshalb außerhalb der Betrachtungen.

Focus, AutoFocus und AutoRaid

Focus ist eine Theorie zu verteilten interaktiven Systemen mit der Strukturierung in Komponenten [BS01]. Die Attraktivität dieser Theorie gegenüber anderen mathematischen Konzepten, wie etwa CSP, liegt in der Übereinstimmung von Verhalten und Beobachtung. Das mathematische Modell einer Komponente in Focus entspricht also ihrem von außen beobachtbarem Verhalten, welches durch den Empfang und das Senden von Nachrichten gegeben ist.

Auf der Basis des Focus Modells ist eine erweiterte Theorie für multifunktionale Softwaresysteme entstanden [Bro03a, Bro05]. Diese Theorie umfasst nicht nur Aspekte des Requirements Engineering bei der Beschreibung der Systemfunktionalität aus Nutzungssicht, sondern auch den Zusammenhang zu logischen Komponentenarchitekturen. Sie schafft damit die Grundlage für die Entwicklung von neuen Modellierungstechniken und Werkzeugen. Die in dieser Arbeit entwickelten Konzepte stützen sich daher auf diese Theorie ab. Für eine Übersicht der dafür wesentlichen Grundlagen sei der Leser an dieser Stelle auf das nachfolgende Kapitel verwiesen.

Auf der Basis von Focus wurde nun an der Technischen Universität München AutoFocus [HMR+98,Slo98] entwickelt. Was zunächst als prototypisches Spezifikationswerkzeug gedacht war, welches Teile der Focus Notation verwendet, steht heute bereits als AutoFocus 2 [AF205] zur Verfügung. Mit AutoFocus können grafisch verteilte Systeme spezifiziert werden, wobei statische Sichten (Systemstrukturdiagramme und Datendefinitionen) und dynamische Sichten (Zustandsdiagramme und Ereignisabfolgen) integriert sind.

Ebenfalls an der Technischen Universität München wurde im Rahmen eines Softwaretechnikpraktikums eine Werkzeugimplementierung namens AutoRaid umgesetzt [RAI05]. Dort wurden Ansätze des Requirements Engineering in das Modellierungswerkzeug AutoFocus 2 integriert. Mit AutoRaid werden Anforderungen nicht wie bei klassischen Anforderungsmanagementwerkzeugen nur erfasst und verwaltet, wie etwa bei DOORS

[DOO05], sondern können modellbasiert analysiert werden. Sie werden dort anhand verschiedener Sichten strukturiert und bereits Elementen des Systemmodells zugeordnet. Hiermit soll der Übergang zum Design verbessert werden. AutoRaid besitzt eine grafische Benutzerschnittstelle und wurde ursprünglich zur Spezifikation von eingebetteten Systemen entwickelt.

Mithilfe von AutoRaid können unstrukturierte Anforderungen, die aus beliebigen projektrelevanten Quellen, zum Beispiel aus dem Lastenheft oder aus Protokollen, gewonnen werden, systematisch strukturiert werden. Auf höchstem Abstraktionsniveau werden so genannte „Business Requirements“ von „Application Requirements“ unterschieden. Jedes Application Requirement muss letztlich mindestens einem Business Requirement zugeordnet sein.
AutoRaid bietet vier Sichten, um funktionale Anforderungen zu strukturieren: „Use Cases“ und „Constraints“ hinsichtlich Architektur, Betriebsart („Mode“) und Datentyp. Neben dieser Klassifizierung können funktionale Anforderungen auch hierarchisch strukturiert werden.

Architekturelle Bedingungen motivieren die Modellierung von Systemkomponenten. Analog wird die Modellierung von Zuständen durch Betriebsmodi und die Definition von Datentypen durch Datentypanforderungen motiviert. Alle diese Modellelemente, also Komponenten, Zustände und Datentypen, können wiederum mit Application Requirements assoziiert werden. Eine Anforderung kann mit bestimmten Sichten assoziiert werden und Elemente der Komponentenarchitektur motivieren.

Es werden also zwei Arten von Beziehungen unterstützt. Durch so genannte „Motivation“ und „Association“ werden funktionale Anforderungen und entsprechende Modellelemente in Beziehung gesetzt. Diese Art von Beziehungen ist begründet durch die Verfolgbarkeit von Anforderungen. Diese Art von Beziehung ist sehr grob und drückt nur die Existenz des Bezugs an sich aus. Wir bezeichnen diese Beziehung daher als *Referenz*. Durch die Klassifizierung der Anforderungen durch die vier unterschiedlichen Sichten sind die möglichen Referenzbeziehungen eingeschränkt. Eine Datentypdefinition kann beispielsweise nicht von einer funktionalen Anforderung referenziert werden, die durch einen Use Case formuliert ist.

Die zweite Art ist eine im Requirements Engineering Modell selbst verankerte Beziehung. Funktionale Anforderungen werden nicht nur Business Anforderungen zu- beziehungsweise untergeordnet. Sie selbst können auch hierarchisch strukturiert werden. Wenn es sinnvoll erscheint, eine geforderte Funktionalität f in mehrere Teilfunktionalitäten $f_1, \ldots, f_n$ aufzuteilen, werden entsprechende Teile von f extrahiert und die f_i der Anforderung f untergeordnet. Implizit wird so eine *Teilfunktion*sbeziehung zwischen den f_i und f hergestellt.

Systems Modeling Language (SysML)

Die SysML [Sys05] entstand im Rahmen einer Arbeitsgruppe, die unter anderem aus Vertretern der Industrie, Behörden und Werkzeughersteller gebildet wurde, um die UML für das Requirements Engineering standardmäßig zu erweitern. Der auf der UML 2.0 [OMG04] basierende Vorschlag wurde bereits in der dritten überarbeiteten Version bei der OMG [OMG05] eingereicht.

Im Rahmen dieser Arbeit interessieren vor allem die Erweiterungen zur Modellierung von Anforderungen. Dies betrifft im Wesentlichen die neu eingeführte Diagrammart „Anforderungsdiagramm“ sowie die dort verankerten Modellierungsmöglichkeiten von Beziehungen. Anforderungen werden durch den Stereotyp `«requirement»` gekennzeichnet, wobei bei Bedarf zusätzliche Anforderungssubtypen durch Ableitung von `«requirement»` gebildet werden können.
Die Definition von Subtypen ist nicht zu verwechseln mit der Dekomposition von Anforderungen in Subanforderungen. Mit dieser „containment“ Beziehung können Anforderungsbäume aufgebaut werden, analog den hierarchisch strukturierten Anforderungen in AutoRaid. Die Strukturierung von funktionalen Anforderungen entspricht damit auch hier einer implizit gegebenen *Teilfunktion*sbeziehung.
Darüber hinaus können Anforderungen selbst in Beziehung gesetzt werden, und zwar durch `«derive»`. Damit wird die Ableitung oder die Generierung einer neuen Anforderung aus einer gegebenen Anforderung zum Ausdruck gebracht. Mit `«satisfy»` kann ebenso wie in AutoRaid eine Beziehung zu Architekturelementen hergestellt werden. Schließlich wird mit `«verify»` der Bezug einer Anforderung zu Testfällen festgelegt.

Neben den Aktivitäten zur Entwicklung der SysML gibt es noch eine Reihe weiterer Ansätze, die Einsatzfähigkeit der UML für ein modernes Requirements Engineering zu verbessern. [Ber04b] ist ein Beispiel, bei dem versucht wird, funktionale und nichtfunktionale Anforderungen nicht isoliert voneinander zu betrachten, sondern Nutzungsfälle zusammen mit möglichen Risiken („hazards“) in ein Modell zu integrieren. Auch hierzu wird ein erweiterter Satz von Beziehungstypen in Form von Stereotypen vorgeschlagen, welche die Zusammenhänge zwischen funktionalen Anforderungen, nichtfunktionalen Anforderungen und Risiken charakterisieren.

2.2 Architekturen

Die Architektur eines Softwaresystems spielt aus Nutzungssicht zunächst eine untergeordnete Rolle, denn das Nutzungsverhalten des Systems ist prinzipiell unabhängig von der gewählten strukturellen Organisation. Da aber jeder Architekturstil Vorteile für bestimmte Domänen oder Anwendungen gegenüber anderen Architekturen aufweist, betrachten wir dennoch einige ausgewählte Architekturen. Dabei werden die typischen Eigenschaften der Relationen zwischen den jeweiligen Systementitäten untersucht.

Architekturelle Stile

Datenflussgetriebene Architekturen, wie zum Beispiel eine Pipe & Filter Architektur, zeichnen sich durch zeitlich unabhängige, sequenzielle Transformation von Datenströmen aus. Eine Transformation kann aber erst dann beginnen, wenn die vorangegangene abgeschlossen ist. Es besteht dort also ein kausaler Zusammenhang.
Filterkomponenten sind Blackboxes, die leicht ersetzbar und beliebig kombinierbar sind, solange ein gemeinsames Datenformat eingehalten wird. Betrachtet man die Kombination zweier Filter, so wird aufgrund der inhärenten Filtereigenschaft das Verhalten der Kombination in der Regel ein anderes sein als die Summe der isolierten Filterverhalten.

Wenn weniger die Datenflüsse, sondern (eine große Menge von) Daten an sich im Zentrum stehen, sind **datenzentrierte Architekturen**, wie Blackboards oder Datenbanken, vorzufinden. Ein typisches Verhaltensmerkmal solcher Softwaresysteme ist das Auslösen von Aktionen bei Änderung von gemeinsam zugreifbaren Daten. Bei Blackboardsystemen erfolgt eine Benachrichtigung an registrierte Systementitäten und bei Datenbanken eine Triggerung von bestimmten Funktionen.
In der Kombination der Systemfunktionen führt das Vorhandensein beziehungsweise die Aktivierung von Triggerfunktionen zu einem geänderten Verhalten.

Bei **aufruforientierten Architekturen** ist der Ablauf von Aktionen festgelegt. Eine Systementität kann ihre Funktion nicht (sinnvoll) ohne das Ergebnis von anderen erbringen. Beispielsweise bei RPC-basierten Client-Server- oder objektorientierten Architekturen wird dies durch eine synchrone Kommunikation erreicht (Methodenaufruf). Das Einfügen von benötigten Funktionalitäten, wie bereits bei der Modellierung mit UML Stereotyp `«include»` erwähnt, beziehungsweise die Nutzung von Funktionalitäten sind dort typische Verhaltenbeziehungen.

Auch eine **Schichtenarchitektur** ist grundsätzlich aufruforientiert. Sie besteht aus einer Hierarchie mehrerer Schichten, wobei eine Schicht von den Details der darunter liegenden Schicht abstrahiert. Eine Schicht selbst ist ein Dienst, der aber nicht nur die Rolle des Diensterbringers, sondern auch die Rolle des Dienstnutzers einnimmt (siehe auch „The Janus-Approach“ [Bro05]). Außer der untersten Schicht besitzt demnach jede Schicht zwei Schnittstellen, eine Exportschnittstelle (der angebotene Dienst) und eine Downward-Schnittstelle, welche den Dienst der darunter liegenden Schicht nutzt.
Mit typischen Schichtenmodellen, wie etwa den Schichten des OSI-Modells [Tan89], ist der Begriff des Protokolls eng verbunden. Ein Protokoll ist die – für den Dienstnutzer nicht sichtbare – Ausführung eines Diensts innerhalb derselben Schicht. Zur Erbringung eines Diensts sind aber stets zwei Schichten beteiligt, die durch die Spezifikation einer Interaktionsschnittstelle komponiert werden. Im Falle des OSI-Schichtenmodells bietet die Transportschicht etwa die Teildienste `T-CONNECT`, `T-DISCONNECT` und `T-DATA` mit den Operationen `request`, `indication`, `response` und `confirm` an, welche dafür die Teildienste `N-CONNECT`, `N-DISCONNECT` und `N-DATA` der darunter liegenden Vermittlungs-

schicht nutzen.
Eine Schicht stellt Funktionen zur Verfügung, die von der darüber liegenden Schicht genutzt werden. Bei Schichtenarchitekturen sind typische Funktionsbeziehungen derart, dass Funktionen andere Funktionen benötigen und nutzen, um einen Dienst zu erbringen. Aus Sicht der darunter liegenden Schicht schaffen Funktionen erst die Voraussetzung für die darüber liegenden, abstrahierenden Funktionen.

Bei den obigen Stilen hängen die Systementitäten mehr oder weniger eng über Daten, Datenflüsse oder Ablaufverhalten zusammen. Als letzten Stil betrachten wir den der **losen Kopplung**. Hier können Systementitäten weitgehend unabhängig voneinander arbeiten, die über den Austausch von Nachrichten interagieren. Als Beispiel seien asynchrone Client-Server- oder ereignisbasierte Architekturen genannt. Dem Senden eines Ereignisses an entsprechende Empfänger folgt eine beliebige Behandlung desselben.
Systementitäten können hier beliebig ausgetauscht werden. In der Kombination von zwei Systementitäten ist eine Abhängigkeit dadurch gegeben, dass bestimmte Ereignisse durch eine Systementität angemeldet sein müssen, für welche sich die andere registriert. Eine Entität schafft also (durch die Bereitstellung von Ereignissen) die Voraussetzung für eine andere.

Architecture Definition Languages (ADL)

Prominente Vertreter von ADLs sind die Sprachen Wright [All97,ADG98], Darwin [MK96], Rapide [Luc96] oder Unicon [SDZ96]. Mit den ADLs können Designeigenschaften eines Systems im Sinne einer Architektur, die als Komposition von Komponenten strukturiert ist, beschrieben werden. Die Komponenten interagieren dabei über so genannte „Konnektoren“. Damit kann eine Reihe unterschiedlichster Aspekte eines Softwaresystems beschrieben werden. Uns interessiert hier jedoch speziell das funktionale Verhalten.

Neben dem statischen Verhalten eines Systems ist der Schwerpunkt der oben genannten ADLs die *Systemdynamik*, also die sich während eines Ablaufs ändernde Architektur (Rekonfigurationen) und das daraus resultierende Systemverhalten.

Die Unterschiede der ADLs sind letztlich im Detail zu finden. Mit Wright etwa wird die Systemdynamik und das statische Verhalten durch einen Formalismus beschrieben, bei dem beide Aspekte isoliert voneinander betrachtet werden können.
In Wright wird ein System als Menge von Komponententypen und Konnektortypen beschrieben. Mit der Deklaration einer Menge von Instanzen dieser Typen, zusammen mit einer Vorschrift wie diese Instanzen miteinander verbunden werden, wird eine Konfiguration definiert. Darwin beschreibt hingegen Komponententypen als Schnittstelle, bestehend aus einer Sammlung von Diensten, die entweder angeboten („provided“) oder benötigt werden („required“). Eine Konfiguration wird dort ebenfalls durch Instanzen der Komponententypen definiert. Allerdings werden hier die *required* mit den *provided* Diensten verbunden. Komponententypen sind in Rapide ebenfalls als Schnittstellen definiert, allerdings als Menge von Kommunikationsereignissen. Die Interaktion zwischen

Komponenten wird durch das Verbinden von gemeinsamen Ereignissen oder kausalen Beziehungen zwischen Ereignissen erreicht.
Das Verbindungsmodell von Darwin und Rapide ist asymmetrisch, das von Unicon und Wright dagegen symmetrisch. Interaktionsmuster können im symmetrischen Fall unabhängig von einer *provider*-Komponente beschrieben werden. Die Menge von Interaktionstypen (etwa *filter* oder *pipe*) sind allerdings bei allen ADLs vorgegeben.

Die Beschreibung von Verhalten und des Zusammenhangs von Einzelverhalten ist sehr detailliert und aus Nutzungssicht nicht brauchbar. Das mit den ADLs spezifizierte Verhalten der Komponenten und Konnektoren kann analysiert werden, indem in der Regel durch Simulation die Erfüllung von (temporallogischen) Systemeigenschaften anhand von Traces (Ablauf des Modells) verifiziert wird.
Interaktionen werden durch kommunizierende, also Nachrichten austauschende, Komponenten modelliert. Die Tatsache, dass – insbesondere auch ungewollte – Interaktionen nicht nur durch Kommunikationsbeziehungen zustande kommen, ist für die ADLs aufgrund der Sichtweise (logische Komponentenarchitektur) und der Phase im Entwicklungsprozess (Entwurf) nicht relevant.

Wir greifen allerdings zwei für diese Arbeit interessierende Konzepte heraus: Die Systemdynamik mit wechselnden Architekturen ist das Resultat der Verfügbarkeit von Diensten. Aus Nutzungssicht betrachtet bedeutet dies, dass die Verfügbarkeit von Funktionen die Voraussetzung für andere Funktionen ist. Wenn wir nicht nur dynamische, sondern auch statische Konfigurationen von Funktionen in Betracht ziehen, stellt auch die Zugehörigkeit zu einer gemeinsamen Konfiguration eine mögliche Art der Beziehung dar.
Die Verbindung von *required* mit *provided* Diensten spielt aus Nutzungssicht ebenfalls eine wichtige Rolle, insofern sich Funktionen zur Erbringung ihrer Dienste auf andere Funktionen abstützen und damit benötigen.

2.3 Dienstbasierte Modellierung

Business Prozesse und Web Services

Bei dem klassischen Client-Server Ansatz [OHE94] des Webs greifen Client-Funktionen auf Daten mithilfe von Server-Funktionen zu. Web Services hingegen ermöglichen die Kommunikation zwischen Maschinen, und zwar den Austausch von Daten und Funktionalität, wie es zum Beispiel schon als Konzept bei CORBA vorliegt [OHE96]. Die klassische Aufruf-Antwort-Beziehung zwischen Nutzer und Anbieter ist aber auch dort zu beobachten.
Die Funktionalitäten von Web Services resultieren aus der Zusammenschaltung verteilter Komponenten. Sie abstrahieren dabei von einzelnen Komponentenschnittstellen und stellen wohl definierte Ergebnisse zur Verfügung [STK02]. Die Web Services selbst können

wiederum zu höherwertigen Funktionen komponiert werden. Dafür werden verschiedene Prozessspezifikationssprachen, wie zum Beispiel die *Business Process Execution Language* (BPEL) [TAC+03], angeboten, welche die Beschreibung komplexer Szenarien auf der Basis von Web Services ermöglichen. Die damit festgelegte Steuerung und das Zusammenspiel mehrerer Web Services wird ferner als „Web Service Orchestrierung“ bezeichnet [HL04].
Neben der Verwendung einfacher Aktivitäten, wie dem Aufruf eines Web Services, dem Empfang oder dem Senden einer Antwort, können komplexere Web Services mit so genannten „strukturierten Aktivitäten“ gebildet werden. Dabei kann eine feste Reihenfolge der Aktivitäten (`Sequence`) oder eine parallele Ausführung (`Flow`) definiert werden. Außerdem kann eine Aktivität erst nach Eintritt eines erwarteten Ereignisses (`Pick`) ausgeführt werden. Mit `Switch` und `While` werden bedingte beziehungsweise wiederholte Ausführungen beschrieben.
Bei der Konzeption von Web Services beziehungsweise von Business Prozessen auf der Basis von Web Services liegt der Schwerpunkt auf dem Laufzeitverhalten. Durch die Komposition von Web Services sind die Abhängigkeiten dort durch Beziehungen wie „uses“ oder „provides“ charakterisiert. Bei der Orchestrierung kann auf der anderen Seite das Verhalten von Web Services, genauer deren Ausführung, ebenso in kausalem Zusammenhang stehen.

Häufig werden Business Prozesse mit der *Integration DEFinition language 0* (*IDEF0*) [Nat93] Technik modelliert. Dies ist ein allgemeiner grafischer Ansatz zur Beschreibung von Systemen. Die zur Verfügung gestellten Modellierungseinheiten sind Funktionen (als so genannte *Boxes* dargestellt) sowie Daten und Objekte, welche die Funktionen in Beziehung setzen (als *Arrows* dargestellt). Funktionen selbst können wiederum hierarchisch strukturiert werden, das durch eine Reihe von gegenseitig referenzierenden Diagrammen erreicht wird.
Die Modellierung einer Business-Funktion besteht in IDEF0 aus so genannten *inputs*, *outputs*, *controls* und *mechanisms*. Durch „inputs“ werden die Auslösemechanismen (trigger) angegeben, welche zur Ausführung der Funktion führen. Mit den „controls“ werden die Bedingungen angegeben, die von der Funktion benötigt werden, um korrekte Ausgaben zu erzeugen. Die Ausgaben „outputs“ bezeichnen die von der Funktion bereitgestellten Daten oder Objekte, die wiederum als „control“ oder „input“ für weitere Funktionen dienen können. Bei den „mechanisms“ wird unterschieden in Hilfsquellen, welche die Ausführung der Funktion unterstützen, und in so genannte *calls*. „calls“ erlauben die gemeinsame Nutzung von spezifischeren Modellen oder Funktionen. Die aufgerufene Funktion stellt Details für die aufrufende Funktion zur Verfügung.
Wir greifen als abstrakte Beziehungstypen hier die Idee der Kontrolle und der gemeinsamen Ressourcennutzung auf. Auch das Triggern einer Funktion stellt einen interessanten Beziehungstyp dar.

Dienstbasierte Spezifikation von Komponenten

Das Konzept eines Diensts findet nicht nur Anwendung im Bereich der Web-Services. Dienste werden ferner als Mittel zur Spezifikation reaktiver Systeme auch im Bereich der eingebetteten Systeme eingesetzt. Beim dienstbasierten Ansatz steht beim Aufbrechen einer Systemfunktionalität weniger der strukturelle Aspekt einer Komponentenarchitektur, sondern vielmehr eine Verhaltensarchitektur im Vordergrund. In [SS03a, SS03b] wird ein Dienst als (wiederverwendbares) Blackboxverhalten beschrieben, welcher eine syntaktische Schnittstelle besitzt, dessen Verhaltenspezifikation aber abhängig von bestimmten Annahmen über die Umgebung des Diensts ist. Diese Dienstumgebung wird durch so genannte „needed services“ spezifiziert. Komponenten bieten entsprechend bestimmte Dienste als „provided services“ an.

Um der bei der Kombination von Diensten auftretenden Problematik der Feature Interaction zu begegnen, wird in [SS03a] eine so genannte „Kompatibilitätshierarchie“ eingeführt. Ein kompatibler Satz von Diensten kann ohne (ungewollte) Feature Interaction kombiniert werden.
Man spricht dort von einer „konsistenten“ Kombination von Diensten, wenn die Domänen[1] der isoliert betrachteten Dienste Teilmengen der Kombinationsdomäne sind. Wenn die Schnittmenge von Dienstdomänen leer ist, existiert ein konsistent kombinierter Dienst und man spricht von einer „schwachen Kompatibilität“ der Dienste. Existiert eine Komponente, die den kombinierten Dienst zur Verfügung stellt, wird dies als „starke Kompatibilität“ bezeichnet. Die Kombinierbarkeit von Diensten lässt sich darüber hinaus noch an den Gegebenheiten der syntaktischen Schnittstellen festmachen. Eine so genannte „Ausgabeschnittstellenkompatibilität“ ist gegeben, wenn die Ausgabekanäle paarweise disjunkt sind. Wenn zusätzlich noch die Eingabekanäle paarweise disjunkt sind, liegt eine „Schnittstellenkompatibilität“ vor.

Der in [SS03a, SS03b] skizzierte dienstbasierte Entwicklungsprozess setzt auf Use Case Spezifikationen auf, mithilfe derer Dienste zunächst isoliert definiert werden. Die Kombination verschiedener Dienste wird über die Zusammenhänge (`needed`) der Dienste im Rahmen einer „Dienstarchitektur“ spezifiziert. Beim „Deployment“ werden dann die Dienste auf eine Menge von Komponenten abgebildet, die diese dann zur Verfügung stellten (`provided`).
Der Schritt von isolierten Diensten hin zu einer Dienstarchitektur mit `needed`-Beziehungen zwischen den Diensten erscheint hier sehr groß und ist bereits die unmittelbare Vorstufe zur Abbildung auf eine logische Komponentenarchitektur. Es bleibt offen, wie man zu diesen Dienstabhängigkeiten gelangt. Der Beitrag dieser Arbeit versucht diese Lücke zu schließen, indem systematisch Beziehungen zwischen Funktionen, und zwar aus

[1] Mit der „Domäne“ einer Funktion wird die Menge der Eingabesequenzen bezeichnet, die zu einem spezifizierten Verhalten der Funktion führt. Die formale Definition ist im nachfolgenden Grundlagenkapitel zu finden.

Nutzungssicht, entwickelt werden. Im Gegensatz zu der grundlegenden Eigenschaft der Kombinierbarkeit von Funktionen sind detailliert spezifizierte Interaktionsschnittstellen zwischen den Funktionen dabei von untergeordnetem Interesse. Gemeinsam verwendete Nachrichten und die Teilung des Zustandsraums eines Systems werden durch Beziehungstypen abstrahiert.

Spontane und adaptive Softwaresysteme

In [Sal02] werden spontane Komponentensysteme als Spezialfall verteilter Komponentensysteme untersucht. Wie bereits im Zusammenhang mit den ADLs beschrieben, zeichnen sich solche Softwaresysteme durch einen hohen Grad an Dynamik hinsichtlich ihrer Verbindungsstrukturen aus. Der Ansatz hier ist, Dienste und Dienstschnittstellen zur Definition einer invarianten logischen Architektur heranzuziehen. Diese logische Architektur wird dann zur Laufzeit, abhängig von den tatsächlich zur Verfügung stehenden Komponenten, auf eine technische Architektur abgebildet.
Um die Güte verschiedener technischer Architekturen zu bewerten, wird zur Beschreibung eine Service Architecture Definition Language (SADL) eingeführt. Letztendlich reduziert sich aber auch hier eine Beziehung zwischen den Funktionen, wie bereits bei den Web Services gesehen, auf einen „needed"- beziehungsweise „provided"- Zusammenhang.

Im Projekt *Mewadis* [MEW05] an der Technischen Universität München werden in Zusammenarbeit mit Industriepartnern Techniken zur Analyse, Modellierung und Validierung für die Entwicklung kontextbezogener Dienste bereitgestellt. Dies schließt Modellierungsmethoden aber auch Vorgehensmethoden ein, um zuverlässige adaptive Dienste entwickeln zu können [DGJW04, DGH+05].
Dort sind Dienste nicht nur durch syntaktische Schnittstellen beschrieben. Neben der semantischen Schnittstellenbeschreibung beinhalten sie dort auch spezielle Diensteigenschaften, zum Beispiel *Qualitiy of Service*-Eigenschaften. Der für diese Arbeit wesentliche Punkt jedoch ist, dass Dienstspezifikationen dort insbesondere auch explizit Beziehungen zu anderen Diensten enthalten.
Eine der Phasen des dort beschriebenen dienstorientierten Entwicklungsprozesses ist die so genannte „Use Case Modelling Phase" [DGP+04a, DGP+04b]. Als Idee ist eine „Logical Service Architecture" als Produkt dieser Phase bereits dargestellt. Sie enthält jedoch keine detaillierten Beziehungstypen und auch keine Semantik von Beziehungstypen. Allerdings wird eine Technik für die Integration von Beziehungen in das CASE Werkzeug AutoFocus vorgeschlagen. Beziehungen sollen als Annotation mit dem Modellelement „Dienst" verbunden werden und im weiteren Verlauf des Entwicklungsprozesses als Parameter für mögliche Modelltransformationen verwendet werden.

Dienstaspekte

In [Rit04, DKMR05] liegt der Schwerpunkt ebenfalls auf dienstorientierten Entwicklungsmethoden. Hier wird die Kombination von Diensten untersucht, sowie so genannte „cross-

cutting“ Features, also Dienste, welche das Verhalten anderer Dienste beeinflussen.

Die untersuchten Beziehungen zwischen Diensten richten sich in [Rit04] auf solche, die den Kontrollfluss beziehungsweise die Kommunikation betreffen. Wie bereits bei den UML Interaktionen diskutiert, werden hier sequenzielle, alternative oder parallele Ausführungen von Diensten untersucht. Neben Aufrufen von Diensten mit („request-response“) und ohne („one-way“) Antwort können Dienste auch verschachtelt sein. Diese Art der Organisation von Diensten bezieht sich auf den Kontrollfluss und ist nicht zu verwechseln mit einer Teilverhaltensbeziehung. Das Verhalten von Diensten kann sich aufgrund einer Unterbrechung durch andere Dienste ändern. Diese Art von Beziehung spielt insbesondere bei der Kombination mit crosscutting Diensten eine Rolle.

Mit dem Konzept von crosscutting Diensten werden aspektorientierte Modellierungstechniken in Sequenzdiagramme eingeführt. Damit wird nicht nur die Beeinflussung des Verhaltens eines Diensts dargestellt, sondern die Modellierung eines übergreifenden Verhaltens von Diensten ähnlich dem Aspektkonzept der aspektorientierten Programmierung. Mit so genannten „join points“ (`Before`, `After`, `Around`, `Instead_of`) können Dienste entsprechend kombiniert werden.
Aus Nutzungssicht sind die Verhaltensänderungen von Funktionen aufgrund kausaler Abhängigkeiten oder Unterbrechungen zu beobachten. Ob eine Funktion *crosscutting* ist, spielt hierbei keine Rolle, eher die dadurch hervorgerufene Verhaltenserweiterung.

In [Rit08] wird die Idee dieser Arbeit der formalisierten Typen von Beziehungen zwischen Nutzungsfunktionen aufgegriffen. Dort wird eine Methode zur Realisierung von Funktionsbeziehungen entworfen, die ebenfalls als Bindeglied zwischen der Anforderungsanalyse und dem Entwurf eines Systems vorgesehen ist. Die Festlegung einer Funktionsbeziehung führt dabei bereits zu einer entsprechend konkreten Änderung der Interaktionsschnittstelle der beteiligten logischen Komponenten.

Feature Komposition

Die „Distributed Feature Composition“ (DFC) ist eine virtuelle Architektur zur Beschreibung von Telekommunikationsdiensten [Zav01, JZ03, BCP+04]. Mit der DFC können Telekommunikationsdienste implementierungsunabhängig durch die Komposition von so genannten „Features“ spezifiziert werden (Feature-orientierte Spezifikation). Features sind dort – analog zur Pipe & Filter Modularität – modulare, funktionale Einheiten, die im Prinzip beliebig sequenziell zusammengestellt werden können und damit eine Basisspezifikation erweitern. Die Basisspezifikation entspricht in der Telefonie einem so genannten „basic call“ oder „direct call“. Ein Telekommunikationsdienst setzt sich demnach stets aus einer Grundfunktionalität und beliebig vielen Features zusammen.

Das primäre Ziel der DFC ist die formale Beschreibung von Telekommunikationsdiensten sowie die Diagnose von Feature Interactions durch Traces. Durch Hinzunahme (`subscribe`) oder Entfernen (`unsubscribe`) von Features wird das Gesamtverhalten geändert. Werden

Features abstrakt als Funktionen betrachtet, so können sie das Verhalten einer Funktion im Sinne einer Erweiterung ändern.

Features ändern aber nicht nur das Gesamtverhalten. Obwohl sie unabhängig voneinander spezifiziert werden und keine direkten Kommunikationsbeziehungen, also keine Interaktionsschnittstellen, definieren, können sie sich trotzdem gegenseitig beeinflussen („Feature Interaction“). Jedes aktivierte Feature kann nämlich Nachrichten (Signale) erzeugen, weiterleiten oder verwerfen und selbstständig Verbindungen aufbauen, annehmen oder auch abweisen. Die indirekten Auswirkungen ergeben sich so aus den (geänderten) Routinginformationen, die ein so genannter „Router“ benutzt, um ein anderes Feature zu signalisieren. Eine Strategie zur Vermeidung von ungewollten Feature Interactions ist eine partielle Ordnung der Features, die dem Router als Regeln zur Verfügung stehen.

In DFC werden unterschiedliche Arten von Feature Interactions klassifiziert, die auch je nach beabsichtigtem Verhalten gewollt oder ungewollt sind. Diese Interaktionstypen sind zwar sehr telekommunikationsspezifisch, die Effekte lassen sich jedoch weitgehend verallgemeinern. Mit `Cancels` wird die Aufhebung einer Funktion ausgedrückt. Die Ersetzung einer Funktion durch eine andere wird durch `Retarget` gekennzeichnet. `Spoofs` und `Hides` beziehen sich auf die Erzeugung beziehungsweise die Absorption von Nachrichten. Das Verhalten einer Funktion wird so gewissermaßen durch eine Filterung von Nachrichten geändert. Der `Drowns`- Interaktionstyp sei hier nur der Vollständigkeit halber erwähnt. Dieser betrifft speziell das Abspielen von Sprachhinweisen und ist deshalb nicht von Interesse.

Feature Interaction

Die Kombination von Diensten oder Funktionen und die damit verbundene Problematik der (ungewollten) Feature Interaction ist Gegenstand einer Reihe von Arbeiten. Die Vermeidung von Feature Interaction ist auch die Motivation einiger der oben beschriebenen Ansätze. Sie bedienen sich formaler Methoden, um entweder per Definition mithilfe einer präzisen Beschreibungstechnik [SS03a, SS03b] Feature Interaction nicht zuzulassen, oder aber durch Simulation und Analyse Feature Interaction zu entdecken und durch geeignete Maßnahmen aufzulösen [Zav01].

Ein ebenfalls auf Verhaltensimulation ausgelegter Ansatz auf der Basis von Coloured Petrinetzen (siehe oben) wird in [LTX01] vorgestellt. Dort werden die Ursachen von Feature Interactions in drei Kategorien eingeteilt: die Nutzung von Funktionen durch andere Funktionen, die gemeinsame Nutzung von begrenzten Ressourcen und die Beeinflussung von Funktionen dadurch, dass Funktionen nicht mehr verfügbar sind oder dass sich ihr Verhalten ändert.

In [SEF04] wird der Begriff der „Feature Interaction“ weiter gefasst, indem auch Beziehungen zu nichtfunktionalen Anforderungen zu so genannten „requirements interaction“

einbezogen werden. Dort wird versucht, allgemein die möglichen Wechselwirkungen zwischen Anforderungen zu klassifizieren, die allerdings unabhängig von funktionalen oder nichtfunktionalen Gesichtspunkten sind: Neben wechselwirkenden oder widersprüchlichen Systemeigenschaften können sich Anforderungen bezüglich des Systemverhaltens beeinflussen. Dies ist zurückzuführen auf die Reduktion gemeinsam genutzter Ressourcen, widersprüchliches Verhalten bei simultaner Aktivierung, auf die Verhinderung der Nutzung oder die Unterbrechung beziehungsweise Aufhebung.

2.4 Konfigurationen und Varianten

Feature-Oriented Reuse Method (FORM)

Mit der „Feature Oriented Domain Analysis“ (FODA) [KCH+90] wird das so genannte „Feature oriented Requirements Engineering“ eingeführt. Mit dieser Methode sollen entscheidende, unverwechselbare Eigenschaften von Softwaresystemen innerhalb einer bestimmten Domäne identifiziert werden. FORM [KKL+98] ist eine Erweiterung für FODA und stellt zusätzlich eine Methode für die Entwicklung von domänenspezifischen Softwarearchitekturen und wiederverwendbaren Komponenten auf der Basis von Feature-Modellen zur Verfügung. Feature-Modelle sind im Wesentlichen *AND/OR* Grafen, mit denen eine Menge möglicher Produkte einer Domäne definiert werden.

Features sind dort hierarchisch strukturiert mithilfe von Kompositions-, Spezialisierungs- und Implementierungsbeziehungen, wobei Features selbst wieder entweder obligatorisch, optional oder alternativ sein können. In [CE00] werden Feature Modelle noch durch weitere Relationen ergänzt. Zusätzlich zur Alternative (exklusives Oder) können dort Features auch durch *nicht-exklusives Oder* in Beziehung gesetzt werden. Mit *requires* und *excludes* wird die unbedingte Abhängigkeit beziehungsweise der gegenseitige Ausschluss von Features modelliert.

Varianten mit UML Use Cases

Vor allem die Entwicklung von Produktvarianten einer Produktlinie verlangt eine adäquate Modellierung der Variabilität. Bei der Beschreibung von Produktvarianten wird dabei oft nur der Aspekt der Wiederverwendung durch die Angabe gemeinsamer und variabler Merkmale (*Features*) betont. Aus Nutzungssicht sind jedoch auch für ein Produkt vor allem funktionale Anforderungen zu bestimmen.

In [vdML02] wird im Zusammenhang mit der UML Modellierung daher der Begriff der Variabilität differenziert in die Beschreibung verschiedener Produktausprägungen und in die Beschreibung unterschiedlicher Abläufe und Interaktionen zur Laufzeit. Zusätzlich zur statischen Modellierung der Produktstruktur, die etwa durch Feature-Modelle

oder Funktionsbäume erreicht wird (siehe Seite 17), wird eine Beschreibung von Verhaltensvarianten hinsichtlich der Produktausprägungen und den Abläufen zur Laufzeit vorgeschlagen. Dazu soll im Wesentlichen das UML Metamodell geändert werden, um *optionale* oder *alternative* Nutzungsfälle im Rahmen von *UML Use Case Diagrammen* und *UML Use Cases* ausdrücken zu können.

Wie oben erwähnt werden MSCs und HMSCs zur ablauforientierten Beschreibung von Nutzungsfällen eingesetzt (siehe Seite 17). In [CWG04, WCG04] wird das Konzept der Variabilität im Zusammenhang mit Sequenzdiagrammen formalisiert und als Erweiterung für (H)MSCs zur Verfügung gestellt. Dazu werden Variantenbeschreibung in MSCs und Variationspunkte in HMSCs integriert. Für jede gewählte Konfiguration des Feature-Modells können so spezifische Nutzungsfälle einer Produktvariante definiert werden.

Funktionsarchitektur

Für die Entwicklung komplexer Steuergerätenetze in Fahrzeugen wurde eine so genannte „Automotive Modeling Language“ (AML) entwickelt [BR01, BR02, vdBBRS02]. Wesentliche Konzepte der AML sind dabei die unterschiedlichen Abstraktionslevels, die von Szenarien bis hin zur Implementation reichen.
Auf dem Abstraktionslevel „Functions“ werden Funktionen hierarchisch zu so genannten „Building Blocks“ strukturiert (Teilfunktionsbeziehung). Diese Building Blocks werden zur Definition von Varianten herangezogen. Auf dem Abstraktionslevel „Functional Network“ werden Abhängigkeiten zwischen den Funktionen definiert, die entweder die Systemkonfiguration oder die Kommunikation zwischen Funktionen (synchrone Ereignisse) betreffen. Für Konfigurationen wird festgelegt, welche Funktionen sich ausschließen, welche Funktionen miteinander kombiniert und in welcher Reihenfolge Funktionen ausgeführt werden müssen.

2.5 Plattformen

Middleware

Für das Management von Diensten, die insbesondere zur Laufzeit dynamisch verfügbar sind, wird eine Reihe von Middleware eingesetzt. Der Zweck des Auffindens und Nutzens der (verfügbaren) Dienste ist dabei allen gemeinsam. Die den Web Services (siehe Seite 28) zugrunde liegende Infrastruktur ist die so genannte *Service-Oriented Architecture (SOA)*: Ein Dienst wird angeboten durch Publizierung, ein Dienstnutzer sucht und findet den Dienst und schließlich fragt der Dienstnutzer den Dienst an und erhält eine Antwort. Dasselbe Prinzip findet man bei Sun Microsystems *Java Intelligent Network Infrastructure (Jini)* [JIN05] und Microsofts *Universal Plug and Play (UPnP)* [UPP05]. Allerdings kommt dort noch zusätzlich der Aspekt der spontanen Vernetzung ins Spiel.

Die Prüfung von Abhängigkeiten hinsichtlich der Verfügbarkeit von benötigten Diensten obliegt bei allen Ansätzen dem Dienstnutzer.
Aus Nutzungssicht ist es zunächst nicht ausschlaggebend, ob die Komposition von Diensten statisch oder dynamisch durch Suchen und Finden erfolgt. Letztlich ist eine potenzielle „benötigt"-Abhängigkeit von Funktionen zu formulieren.

Die *Open Services Gateway Initiative (OSGi)* geht mit ihrer *OSGi Software Plattform* und dem darin spezifizierten *OSGi Framework* weiter. Dienste werden dort in so genannte „Bundles" eingepackt, deren Abhängigkeiten explizit spezifiziert und damit auch überprüft werden können. Die durch die Nutzung von Diensten durch Bundles oder andere Dienste implizierten Abhängigkeiten werden mit `Import-Service` angezeigt. Bei der Installation eines Bundles werden statische Abhängigkeiten vom Framework überprüft. Dafür werden in [OSG03] unter anderem die Bundlezustände `Installed`, `Resolved` und `Active` definiert. Ein Dienst kann nur dann zur Verfügung gestellt werden, wenn seine statischen Abhängigkeiten zu anderen Diensten aufgelöst sind (`Resolved`). Nur wenn er aktiviert wurde (`Active`), kann ein Dienst benutzt werden.
Neben der bereits oben identifizierten „benötigt"-Abhängigkeit, ist damit noch eine Aktivierungsbeziehung erkennbar.

Dienstbasierte Architekturen bieten sich als Grundlage für die Entwicklung multifunktionaler Systeme, wie etwa Infotronic-Systeme in Fahrzeugen, an. In [NP03] wird dabei als Grund nicht nur der hohe Interaktionsgrad betont, sondern dass insbesondere dort als Prämisse des Systementwurfs und der Systementwicklung nicht mehr die physikalische Verbindung der Subsysteme gilt. Subsysteme beziehungsweise Komponenten sind auf der Basis von „shared information" verbunden und dienstbasierte Methoden dienen zur Strukturierung des Informationsaustauschs. In diesem Zusammenhang wird dort ein Dienst beschrieben als eine wohldefinierte Schnittstelle, über die man spezifische Information erhält oder bestimmte Aktionen ausgeführt werden. Ein Dienst zeichnet sich durch eine einfache Interaktion oder einer Sequenz von Interaktionen zwischen „service user" und „service provider" aus.
Bei den Interaktionen geht es im Wesentlichen um den Anstoß einer Aktion oder das Abfragen von Information (`Request`). Interaktionstypen unterscheiden sich lediglich darin, wann und wie Information zur Verfügung gestellt wird. Neben der synchronen Variante gibt es noch die der periodischen Aktualisierung (`Subscribe`) und der asynchronen, nichtperiodischen Benachrichtigung durch Ereignisse.
Um Dienste mit Anforderungen abgleichen zu können, um zum Beispiel mehrere Interaktionsvarianten zu unterstützen, können Eigenschaften von Diensten konfiguriert werden. Im Falle von OSGi ist dies etwa durch Angabe entsprechender Attribut-Wert Paare möglich.

CARTRONIC

Die Firma Bosch hat unter dem Namen CARTRONIC [BSDV97, LFS+01, KFKK03] eine Systematik als Grundlage zur Systemvernetzung in Fahrzeugen entwickelt. Dem hohen Funktionsumfang in Fahrzeugen und der wachsenden Komplexität wird durch Bildung einer hierarchisch organisierten so genannten „Funktionsarchitektur“ begegnet. Durch einen Satz von Regeln zur Strukturierung und Modellierung werden nur wenige Wechselwirkungen über Kommunikationsbeziehungen erlaubt.

Eine CARTRONIC Funktionsarchitektur besteht im Wesentlichen aus atomaren und zusammengesetzten Komponenten. Letztere werden dort auch als Subsystem bezeichnet. Zugang zu jedem Subsystem bildet stets eine so genannte „Eingangskomponente“. Alle Aufträge an ein Subsystem gehen an dessen Eingangskomponente, welche wiederum für die internen Auftragsvergaben verantwortlich ist.

Beziehungen zwischen Komponenten werden ausschließlich durch drei Arten von Kommunikationsbeziehungen formuliert. Damit können Funktionen nur durch eine sehr begrenzte Auswahl an Interaktionsschnittstellen komponiert werden. Mit einem „Auftrag“ (`order`) ist die Pflicht zur Ausführung verbunden oder der Auftraggeber erhält eine „Rückmeldung“, falls der Auftrag nicht ausgeführt werden kann. Eine „Anforderung“ (`request`) ist dagegen lediglich ein Wunsch zur Ausführung. Die letzte Interaktionsform stellt eine „Abfrage“ (`inquiry`) dar. Damit beschaffen sich Komponenten Informationen, wobei die abgefragten Komponenten losgelöst von der Auswertung der gelieferten Informationen sind.

Prozesse eines Betriebssystems

Auch die Prozesse eines Betriebssystems verdeutlichen weitere Funktionszusammenhänge. Betriebssystemprozesse zeichnen sich vor allem durch die in Kombination mit anderen Prozessen auftretenden Nutzungskonflikte bei gemeinsam, simultan genutzten, Betriebsmitteln (Systemfunktionen) aus. Das Betriebssystem verwaltet die gemeinsame Nutzung von Betriebsmitteln und stellt diese fair und kontrolliert zur Verfügung.

Betriebssystemprozesse können dabei entweder unabhängig voneinander sein, zueinander in Beziehung stehen durch Kooperation über Nachrichtenaustausch, oder auf die Zuteilung von benötigten Ressourcen warten, die von einem anderen Prozess belegt sind. Neben der Unabhängigkeit sind also kausale Zusammenhänge sowie die Nutzung gemeinsamer Ressourcen entscheidende Beziehungstypen. Prozesse können erst die Voraussetzung zur Ausführung anderer Prozesse schaffen oder im Gegenteil auch dafür verantwortlich sein, dass Prozesse deaktiviert werden.

2.6 Zusammenfassung

Mit Ausnahme erster Ansätze des modellbasierten Requirements Engineerings und Feature-basierter Produktdefinitionen werden Zusammenhänge und Abhängigkeiten zwischen Funktionen in heutigen Werkzeugen nicht explizit formuliert. Darüber hinaus spielen Beziehungen von Funktionen aus Nutzungssicht nur eine untergeordnete Rolle. Die verhaltensunabhängigen Beziehungen bei der Definition von Produktvarianten legen nur mögliche Konfigurationen eines Produkts fest. Eine Aussage, ob sich dem Benutzer dadurch das gewünschte oder erwartete Verhalten des Systems zeigt, ist so jedoch nur sehr begrenzt möglich.

Auch wenn das Hauptaugenmerk beim Großteil der beschriebenen Ansätze auf Komponentenarchitekturen, Schnittstellen- oder Interaktionsspezifikationen (Protokolle) liegt, lassen sich trotzdem typische Muster von Verhaltensbeziehungen ableiten, auch wenn sie sich erst implizit aus den angebotenen Spezifikationstechniken ergeben.

Aufgegriffene Konzepte

Der prominenteste Zusammenhang aller betrachteten Ansätze ist die hierarchische Strukturierung von Funktionen. Dabei sind zwei Ausprägungen zu unterscheiden: *Teilverhalten* und höherwertiges Verhalten. Teilverhaltensbeziehungen treten bei der Aufteilung von geforderten Funktionalitäten in Teilfunktionalitäten beziehungsweise bei der Einordnung zu übergeordneter Funktionalität auf. Typischerweise ist dies bei funktionalen Zerlegungen mit Funktionsbäumen oder bei der Definition eines Szenarios als Teilverhalten eines Use Cases zu beobachten. Auch AutoFocus 2 mit AutoRaid unterstützt die hierarchische Strukturierung von Use Cases. Die „Containment“ Beziehungen in SysML zur Bildung von Anforderungsbäumen charakterisieren ebenso diesen Typ von Beziehungen wie die mit der AML spezifizierten „Building Blocks“ der Funktionen in Funktionsarchitekturen.

Wenn „höherwertige“ Funktionen auf Grundfunktionen aufbauen, ist das Verhalten einer Grundfunktion kein Teilverhalten einer höherwertigen Funktion. Klassischer Vertreter solcher Konstruktionen sind Schichtenarchitekturen, bei denen Schichten von den Details tiefer liegender Schichten abstrahieren. Die höherwertige Funktion ist dort keine Kombination mehrerer Schnittstellen, sondern sie stellt eine neue Schnittstelle zur Verfügung und *benötigt* dazu eine Menge anderer Funktionen.

Die *Veränderung von Eingaben* für Funktionen ist eine weitere Art der Beeinflussung. Der Zusammenhang von Filterkomponenten in Pipe & Filter Architekturen ist dadurch gekennzeichnet. Aber auch die Erzeugung beziehungsweise die Absorption von Nachrichten, wie sie bei DFC Interaktionstypen beschrieben ist, sind Beispiele dafür.

Die *Änderung des Verhaltens* einer Funktion ist typisch für Feature-orientierte Spezifikationstechniken. Funktionserweiternde Features der DFC sind hierfür zu nennen, wie auch aspektorientierte Methoden zur Beschreibung von „crosscutting“ Funktionen. Eine

Verhaltensänderung wird beispielsweise auch anhand optionalen Verhaltens mittels der «extend» Beziehung in UML Use Case Diagrammen festgelegt.

Die *Schaffung von Voraussetzungen* beziehungsweise der *Entzug von Voraussetzungen* für Funktionen stellen weitere grundlegende Zusammenhänge dar. Die Spezifikation alternativer Abläufe nur bei bestimmten, erfüllten Bedingungen ist etwa bei erweiterten HMSCs beziehungsweise bei der Spezifikation von UML 2.0 Interaktionen möglich. Die Triggerung von Funktionen ist eine Variante dieser Art von Beziehung. Das Auslösen von Aktionen bei Datenbanken oder Blackboardsystemen oder auch die Auslösemechanismen bei IDEF0 Spezifikationen sind Beispiele dafür.
Auch der Zusammenhang, dass eine Funktion andere Funktionen benötigt (siehe oben), kann dieser Beziehungsart zugeordnet werden. Eine „Benötigt"-Beziehung ist die nach der Teilfunktionsbeziehung am häufigsten benutzte. Sie ist gegeben bei notwendigen Funktionen als Kommunikationspartner zum Beispiel bei Client-Server Architekturen, aber auch bei dienstbasierten Komponentenspezifikationen. Bei den ADLs oder der SOA ist die Verfügbarkeit von Funktionen die Voraussetzung für andere Funktionen.
Betrachtet man die Mechanismen des OSGi Frameworks oder der Betriebssystemprozesse, ragt die Aktivierung oder Deaktivierung von Diensten oder Funktionen heraus. Wenn eine Funktion nicht (mehr) zur Verfügung steht, sind die Voraussetzungen für diese Funktion nicht mehr gegeben. Dies kann beispielsweise mit dem break Operator für UML Interaktionen modelliert werden. Die Aufhebung oder Ersetzung einer Funktion ist ebenfalls im Zusammenhang mit DFC Interaktionstypen beschrieben.

Ferner werden oft *kausale Abhängigkeiten* zwischen Funktionen spezifiziert, sei es mit synchronisierten Aktionen bei Petrinetzen oder mit Bedingungen für das Auslösen einer Aktion bei den CSPs. Das sequenzielle Verhalten von Filterkomponenten in Pipe & Filter Architekturen ist darunter ebenso zu sehen wie die mit „sequencing" geordneten Teilverhalten bei Ablaufspezifikationen. Meist wird durch den kausalen Zusammenhang auch von zeitlichen oder realzeitlichen Beziehungen abstrahiert, wie zum Beispiel bei den „occurrance patterns" und „order patterns" zwischen Ereignissen oder Zuständen.

Nicht zuletzt stellt auch der *unbedingte Zusammenhang* von Funktionen eine Abhängigkeit dar, welcher nicht zwangsläufig auf funktionalen Bedingungen beruht. Mit ADLs kann zum Beispiel die Zugehörigkeit einer Menge von Funktionen zu einer Konfiguration festgelegt werden. Auch die Definition von Produktvarianten mit Feature-Modellen kann einen derartigen Zusammenhang widerspiegeln.

Die *Kombination* von Funktionen ist ein aus Nutzungssicht entscheidendes Konzept. Eine ähnliche Idee ist aus den Techniken zur Analyse von UML Modellen mit dem Begriff des „abstrakten Use Case", als ausschließlich zusammengesetzter Use Case, entstanden. Wenige Arbeiten untersuchen die Kombinierbarkeit von Diensten für Komponentenarchitekturen und die Kombinierbarkeit von Features im Zusammenhang mit der Definition von Produktvarianten einer Produktlinie.

Untergeordnete Konzepte aus Nutzungssicht

Weniger interessant aus Nutzungssicht sind architekturelle Eigenschaften, wie zum Beispiel die „Besteht aus“-Beziehungen bei Komponentenstrukturen in Dekompositionsdiagrammen. Überhaupt stehen solche Zusammenhänge, welche rein die Komponentenarchitektur betreffen, im Hintergrund. So setzen etwa die mit CARTRONIC explizit formulierten Kommunikationsbeziehungen bereits eine Komponentenarchitektur voraus. Diese Arten von Beziehungen werden zwar betrachtet, sind aber aus Nutzungssicht zunächst zweitrangig. Darunter fallen auch „angebotene“ Funktionen einer Komponente, wie sie bei den ADLs oder den dienstbasierten Spezifikationen von Komponenten anzutreffen sind. Für die Nutzungsarchitektur sind Dienste eher im Hinblick auf die Diensterbringung von Interesse und weniger als angebotene Dienste einer Komponente (siehe auch Kapitel 1.2 auf Seite 8).

Die Wiederverwendung von Funktionalität ist ebenfalls nicht im Blickpunkt. `«include»` Beziehungen in UML Use Case Diagrammen oder referenzierte Verhaltensspezifikationen wie bei den HMSCs und bei den „Mechanismen“ der IDEF0 dienen im Wesentlichen der Vermeidung von Redundanz. Auch die Subnetzdefinitionen, wie sie beispielsweise bei den (Coloured) Petri Nets verwendet werden, sind aus Nutzungssicht von untergeordnetem Interesse.

Schließlich ist der Zusammenhang einer höherwertigen Funktion zu den Grundfunktionen nicht im Sinne einer neuen, programmierten Funktion zu verstehen. Beispielsweise ergeben sich aus Schleifenformulierungen in Interaktionsspezifikationen oder Orchestrierungen von Web-Services keine zusätzlichen Arten von Beziehungen.

Fazit

Die systematische Modellierung von Funktionsbeziehungen auf der Nutzungsebene wurde bisher kaum untersucht und ist ein interessantes, augenscheinlich neues Forschungsgebiet.

Wie bereits erwähnt ist es aus Nutzungssicht erforderlich, *Kombinationen* von Funktionen zu untersuchen. Entscheidend ist dabei, den Zusammenhang zwischen dem spezifizierten Verhalten einer Funktion und dem Verhalten in Kombination mit anderen Funktionen beschreiben zu können. Dies erfordert nicht nur die Definition des Begriffs der „Kombination“, sondern insbesondere den Begriff der „Kombinierbarkeit“.

Im Rahmen der Arbeit werden die oben dargestellten Konzepte weitgehend berücksichtigt. Eine wesentliche Funktionsbeziehung ist noch zu ergänzen, und zwar die Unabhängigkeit von Funktionen. Zwischen einer explizit festgelegten Unabhängigkeitsbeziehung und keiner definierten Beziehung ist dabei zu unterscheiden. Die Unabhängigkeit von Funktionen wird zum Beispiel angedeutet durch die Festlegung beliebig verschränkten, überlappenden Verhaltens bei Interaktionsspezifikationen. Allerdings ist zu beachten, dass die Unabhängigkeit von Funktionen nicht zwangsläufig wechselseitig ist.

Bei der Analyse der Ansätze fällt das unterschiedliche Abstraktionsniveau der identifizierten Beziehungsarten auf. Eine Funktion benötigt beispielsweise eine andere Funktion, aktiviert eine andere Funktion oder ist die Voraussetzung einer anderen Funktion. Diese Beziehungsarten sind alle intuitiv verständlich und nachvollziehbar. „benötigt“ und „aktiviert“ stellen aber lediglich eine Präzisierung von „ist Voraussetzung“ dar. Dies legt die Klassifizierung der Beziehungsarten nahe, die insbesondere für die systematische Entwicklung von Nutzungsarchitekturen angewendet werden kann. Ebenso wichtig wie die Kenntnis verschiedener Arten von Funktionsbeziehungen ist also auch das Wissen um die Zusammenhänge der Beziehungsarten selbst. Neben der erwähnten Präzisierung schließen sich möglicherweise verschieden definierte Beziehungen zwischen Funktionen aus.

Kapitel 3

Grundlagen

In der Einführung und bei der Diskussion der verwandten Arbeiten werden unter anderem Begriffe wie die Komposition und die Kombination von Funktionen oder das Verhalten von Funktionen verwendet. Die Bedeutungen der Begriffe werden dort jedoch eher exemplarisch gebraucht. Für die Fundierung der im Anschluss vorgestellten Beziehungskonzepte ist allerdings ein tieferes Verständnis erforderlich. In diesem Kapitel werden daher die notwendigen theoretischen Grundlagen beschrieben.

3.1 Nachrichtenströme, Kanäle und Kanalhistorien

Die Konzepte dieser Arbeit erfordern insbesondere eine kompakte Form der Beschreibung des Verhaltens kombinierter Funktionen. Wir greifen daher auf die von FOCUS angebotenen Techniken zurück. FOCUS ist zunächst eine Theorie zur logischen Beschreibung von verteilten Systemen [BS01]. Systeme werden dort aus nebenläufigen, interagierenden Teilsystemen komponiert. Die Teilsysteme tauschen dabei Nachrichten untereinander aus. Obwohl FOCUS ursprünglich nur für die Spezifikation totalen Verhaltens (Komponenten) konzipiert war, eignet sich die FOCUS-Notation aber auch zur Beschreibung partiellen Verhaltens, wie es typischerweise im Kontext von Nutzungsfunktionen beziehungsweise Diensten erforderlich ist. Das mathematische Modell einer Komponente oder eines Diensts entspricht dem extern beobachtbaren Verhalten, welches mithilfe von so genannten „Kommunikationshistorien“ beschrieben wird. Das dafür verwendete zentrale Konzept in FOCUS ist der „Nachrichtenstrom“ (engl. „stream“).

Nachrichtenströme

Nachrichten werden dabei über Kanäle empfangen und gesendet. Ströme repräsentieren daher auch so genannte „Kanalhistorien“. Auf der Grundlage eines einfachen Zeitmodells[1] wird Verhalten in Zeitintervalle aufgeteilt. Eine Historie wird so als Abbildung von Zeitintervallen auf Nachrichtensequenzen beschrieben.

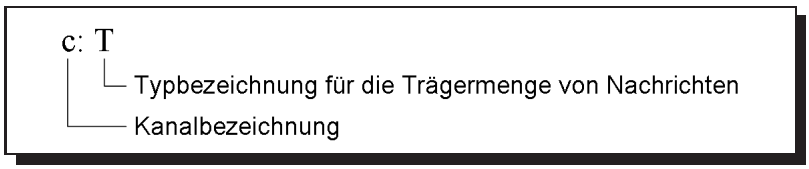
Mathematisch ausgedrückt sei mit T eine Menge von Nachrichten bezeichnet. Die Menge der endlichen Sequenzen von Elementen aus T sei dann mit T^* dargestellt. Mit der Hinzunahme von Zeitinformation gemäß dem oben erwähnten Zeitmodell (unendliche Sequenz von gleich langen Zeitintervallen) ist ein gezeiteter Strom s definiert durch die Zuordnung von Zeitfenstern zu endlichen Nachrichtensequenzen.

$$s : \mathbb{N}_+ \mapsto T^*$$

Mit $s.i$ referenzieren wir auf die Nachrichtensequenz des i-ten Zeitfensters eines Nachrichtenstroms s.

Getypte Kanäle

Ein Kanal ist im Wesentlichen ein *getypter* Bezeichner für einen Nachrichtenstrom, das heißt, nur Nachrichten eines gegebenen Typs werden über diesen Kanal empfangen beziehungsweise gesendet.



Die Menge aller Nachrichten m_i, welche mit T umschrieben wird, wird als *Trägermenge* von T bezeichnet:

$$\mathsf{car}(T) = \{m_1, m_2, \ldots, m_k\}$$

Kanalhistorien

Sei C eine Menge getypter Kanäle,

$$C = \{(c_1\colon T_1), (c_2\colon T_2), \ldots, (c_n\colon T_n)\}$$

[1] Diskreter Zeitbegriff durch so genannte „ticks“.

und $\mathbb{T}$ die Menge aller Nachrichtentypen. Eine Kanalhistorie ist dann gegeben durch die Abbildung h der Kanäle C auf gezeitete Ströme,

$$h : C \mapsto (\mathbb{N}_+ \mapsto \mathbb{T}^*)$$

Mit einer Kanalhistorie wird das Kommunikationsverhalten (Nachrichtenaustausch) auf einer Menge von Kanälen beschrieben. Eine Kanalhistorie ist eine Menge von Nachrichtenströmen mit einem Nachrichtenstrom pro Kanal. In [SS03b] wird dies auch als „System Execution“ bezeichnet.
Auf den Nachrichtenstrom eines bestimmten Kanals c wird mit $h.c$ referenziert.

Kanalhistorien entsprechen nun einer Menge solcher *System Executions*. Im Falle der Kanalmenge C wird die Menge der Kanalhistorien auch mit $\overrightarrow{C}$ bezeichnet:

$$\overrightarrow{C} = \{h_1, h_2, \ldots, h_n\} \quad , mit \quad h_i : C \mapsto (\mathbb{N}_+ \mapsto \mathbb{T}^*)$$

Für die Kanalmenge $C = \{(c_1 \colon T_1), \ldots, (c_n \colon T_n)\}$ wird mit $\mathsf{cid}(C)$ die *Menge der Kanalbezeichnungen* von C angegeben.

$$\mathsf{cid}(C) = \{c_1, c_2, \ldots, c_n\}$$

Wir verwenden cid auch abkürzend dafür, um auf die Kanalbezeichnungen einer gegebenen Kanalhistorie $h \in \overrightarrow{C}$ referenzieren zu können.

$$\mathsf{cid}(h) = \{c : c \in \mathsf{cid}(C) \ \wedge \ h \in \overrightarrow{C}\}$$

Eingabehistorien und Ausgabehistorien

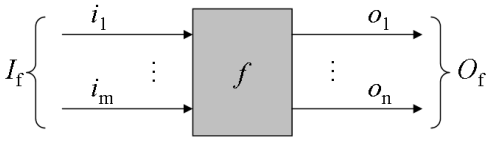


Abbildung 3.1: Syntaktische Schnittstelle

Für eine Menge von Eingabekanälen $I = \{i_1, i_2, \ldots, i_m\}$ bezeichnen wir eine Eingabekanalhistorie, oder kurz: Eingabehistorie, als x mit

$$x : \{i_1, i_2, \ldots, i_m\} \mapsto (\mathbb{N}_+ \mapsto \mathbb{T}^*)$$

Gemäß der oben vereinbarten Schreibweise wird dann die Menge der Eingabehistorien

mit $\overrightarrow{I}$ angegeben:

$$\overrightarrow{I} = \{x : x \in \{i_1, i_2, \ldots, i_m\} \mapsto (\mathbb{N}_+ \mapsto \mathbb{T}^*)\}$$

Analog bezeichnen wir eine Ausgabehistorie mit y und die Menge der Ausgabehistorien mit $\overrightarrow{O}$. Die syntaktische Schnittstelle einer Komponente oder eines Diensts f ist definiert durch die Menge von getypten Eingabekanälen I_f und die Menge von getypten Ausgabekanälen O_f, abgekürzt mit $(I_f \blacktriangleright O_f)$, wie in Abbildung 3.1 grafisch dargestellt.

3.2 Operatoren und Relationen

Die typischen Mengenoperatoren lassen sich auch für Kanalmengen definieren. So ist etwa die Schnittmenge zweier Kanalmengen C und D festgelegt durch die Kanäle mit gemeinsamen Kanalbezeichnungen, wobei der Typ eines gemeinsamen Kanals aus der Schnittmenge der Trägermengen resultiert.

Definition 3.1 (Schnittmenge von Kanalmengen)

Gegeben seien zwei Kanalmengen C und D. Die Schnittmenge der Kanalmengen $C \cap D$ ist definiert als

$$C \cap D \stackrel{\mathsf{def}}{=} \{(x\colon \mathsf{car}(T) \cap \mathsf{car}(U)) : (x\colon T) \in C \wedge (x\colon U) \in D\}$$

□

Die Vereinigungsmenge zweier Kanalmengen lässt sich analog definieren. Die Trägermengen der Kanäle mit gleichen Kanalbezeichnungen werden dabei jeweils vereinigt.

Definition 3.2 (Vereinigungsmenge von Kanalmengen)

Gegeben seien zwei Kanalmengen C und D. Die Vereinigungsmenge der Kanalmengen $C \cap D$ ist definiert als

$$\begin{array}{rcll} C \cup D \stackrel{\mathsf{def}}{=} & \{(x\colon \mathsf{car}(T) \cup \mathsf{car}(U)) & : & x \in \mathsf{cid}(C) \cap \mathsf{cid}(D)\} \\ \cup & \{(x\colon T) \in C & : & x \in \mathsf{cid}(C) \backslash \mathsf{cid}(D)\} \\ \cup & \{(x\colon U) \in D & : & x \in \mathsf{cid}(D) \backslash \mathsf{cid}(C)\} \end{array}$$

□

Auch die Teilmengenrelation lässt sich leicht auf Kanalmengen erweitern. Wir verwenden hierfür allerdings den Begriff des „Subtyps einer Kanalmenge“.

Definition 3.3 (Subtyp einer Kanalmenge)

Eine Kanalmenge C ist Subtyp einer Kanalmenge D, wenn in D alle Kanalbezeichnungen von C enthalten sind und die Trägermenge jedes Kanals von C eine Teilmenge der Trägermenge eines Kanals von D ist:

$$\begin{aligned} & \mathsf{cid}(C) \subseteq \mathsf{cid}(D) \\ \wedge \quad & \forall\ (x\colon T) \in C : \exists\ (x\colon U) \in D \ \mathit{mit}\ \ \mathsf{car}(T) \subseteq \mathsf{car}(U) \end{aligned}$$

Wir schreiben dann $C \subseteq D$. □

Wenn bei zwei Schnittstellendefinitionen die Eingabekanalmengen und die Ausgabekanalmengen jeweils in einer Subtyprelation stehen, verwenden wir auch dann den Begriff des „Subtyps“.

Definition 3.4 (Subtyp einer Schnittstelle)

Gegeben seien zwei Schnittstellen $(I_1 \blacktriangleright O_1)$ und $(I_2 \blacktriangleright O_2)$. Die Schnittstelle $(I_1 \blacktriangleright O_1)$ ist genau dann Subtyp der Schnittstelle $(I_2 \blacktriangleright O_2)$, wenn gilt:

$$I_1 \subseteq I_2 \text{ und } O_1 \subseteq O_2$$

Wir schreiben dann $(I_1 \blacktriangleright O_1) \subseteq_{IF} (I_2 \blacktriangleright O_2)$. □

Operatoren und Relationen auf Kanalhistorien

In [BS01] wird eine Reihe von Operatoren und Relationen auf Nachrichtenströmen definiert. Der Einfachheit fassen wir hier die für diese Arbeit notwendigen Notationen kurz zusammen:

- Um die **Länge** eines Nachrichtenstroms s auszudrücken, wird der Operator $\#$ verwendet. Die Länge von s wird demgemäß als $\#s$ angegeben.
- Auf die ersten n Nachrichtensequenzen des Nachrichtenstroms s wird mit $s{\downarrow_n}$ referenziert und als **Präfix** von s bezeichnet, welches das Verhalten bis zum Zeitpunkt n charakterisiert. Für $n = \infty$ ist das Präfix der Nachrichtenstrom s selbst. Den leeren Nachrichtenstrom $\langle\,\rangle$ erhält man bei $n = 0$.
- Der **Filteroperator** Ⓢ kann bestimmte Nachrichten aus einem Nachrichtenstrom herausfiltern. Aus einem Nachrichtenstrom s erhält man mit M Ⓢ s einen Nachrichtenstrom, in dem alle Nachrichten entfernt sind, die nicht in M enthalten sind.

- Die **Zeitabstraktion** eines Nachrichtenstroms s stellt eine reine Nachrichtensequenz dar und ist formal definiert als $\overline{s} \stackrel{\mathsf{def}}{=} \mathbb{T}\,ⓢ\,s$. Mit anderen Worten, die Zeitinformation („ticks“) wird ausgefiltert.
- Schließlich lässt sich das Zeitfenster oder die Zeiteinheit $\mathsf{tm}(s,k)$ angeben, in der die k-te Nachricht eines Nachrichtenstroms s vorkommt. Der so genannte „time stamp“ ist formal definiert als $\mathsf{tm}(s,k) \stackrel{\mathsf{def}}{=} min\{j \in \mathbb{N}_+ : \#\overline{s\downarrow_j} \geq k\}$.

Mithilfe des „time stamp“ Operators lässt sich für einen Nachrichtenstrom leicht das Zeitfenster der ersten und der letzten übertragenen Nachricht ausdrücken.

Definition 3.5 (Erste und letzte Nachricht eines Nachrichtenstroms)

Das Zeitfenster der ersten übertragenen Nachricht in einem Nachrichtenstrom s wird mit $\mathsf{ftm}(s)$ *bezeichnet und ist definiert als*

$$\mathsf{ftm}(s) \stackrel{\mathsf{def}}{=} \mathsf{tm}(s,1)$$

Das Zeitfenster der letzten übertragenen Nachricht in einem Nachrichtenstrom s wird mit $\mathsf{ltm}(s)$ *bezeichnet und ist definiert als*

$$\mathsf{ltm}(s) \stackrel{\mathsf{def}}{=} \mathsf{tm}(s, \#\overline{s})$$

□

Diese Definitionen für Nachrichtenströme lassen sich verallgemeinern und auf Kanalhistorien übertragen.

Definition 3.6 (Erste und letzte Nachricht einer Kanalhistorie)

Das Zeitfenster der ersten übertragenen Nachricht in einer Kanalhistorie x wird mit $\mathsf{ftm}(x)$ *bezeichnet und ergibt sich aus dem Minimum über alle Nachrichtenströme in x.*

$$\mathsf{ftm}(x) \stackrel{\mathsf{def}}{=} min\{\mathsf{ftm}(x.c) : c \in \mathsf{cid}(x)\}$$

Das Zeitfenster der letzten übertragenen Nachricht in einer Kanalhistorie x wird mit $\mathsf{ltm}(x)$ *bezeichnet und ergibt sich aus dem Maximum über alle Nachrichtenströme in x.*

$$\mathsf{ltm}(x) \stackrel{\mathsf{def}}{=} max\{\mathsf{ltm}(x.c) : c \in \mathsf{cid}(x)\}$$

□

Schließlich lässt sich auch die Definition des Präfixes einer Kanalhistorie aus der eines Nachrichtenstroms ableiten.

Definition 3.7 (Präfix einer Kanalhistorie)

Das Präfix einer Kanalhistorie $x \in \overrightarrow{C}$ wird analog mit $x{\downarrow_n}$ bezeichnet und umfasst die ersten n Nachrichtensequenzen der Nachrichtenströme aller Kanäle in x. Für alle $n \in \mathbb{N}$ und $c \in \mathsf{cid}(C)$ gilt:

$$(x{\downarrow_n}).c \stackrel{\mathsf{def}}{=} (x.c){\downarrow_n}$$

Mit $x{\uparrow_n}$ wird entsprechend der Rest von x bezeichnet. $x{\uparrow_n}$ ist eine Kanalhistorie ohne die ersten n Nachrichtensequenzen der Nachrichtenströme aller Kanäle in x. □

Die Beschreibung einiger der in den folgenden Kapiteln dargestellten Beziehungen zwischen Funktionen erfordert die Charakterisierung bestimmter Muster der Eingabehistorien. Dazu werden im Anschluss Operatoren für die Überlappung von Nachrichten und die zeitliche Verschiebung von Kanalhistorien eingeführt.

Überlappung von Nachrichten in Strömen und Kanalhistorien

Die Überlappungsmöglichkeiten der Nachrichten von Eingabehistorien und Ausgabehistorien sind wesentliche Eigenschaften einer Beziehung zwischen Funktionen und helfen letztlich, die Kombinierbarkeit von Funktionen zu analysieren.

Wir legen zunächst fest, was wir unter der Überlappung der Nachrichten zweier gezeiteter Nachrichtenströme verstehen. Da ein Nachrichtenstrom eine Sequenz von endlichen Nachrichtensequenzen ist, untersuchen wir erst die Bedeutung der Überlappung von Nachrichtensequenzen.
Wir verwenden im Folgenden für den Überlappungsoperator das Symbol $\oplus$.

Definition 3.8 (Überlappung von Nachrichten in Sequenzen)

Gegeben seien zwei Nachrichtensequenzen s und t. Der Operator

$$\oplus \in \mathbb{T}^* \times \mathbb{T}^* \mapsto \wp(\mathbb{T}^*)$$

wird dazu benutzt, die Menge der Nachrichtensequenzen zu erzeugen, welche aus der Überlappung der Nachrichten von s und t hervorgeht, bei der die Reihenfolge der Nachrichten beider Sequenzen erhalten bleibt.
Wir bezeichnen $s \oplus t$ kurz als die Überlappungsmenge *der beiden Nachrichtensequenzen.*

□

Angenommen die Nachrichtensequenzen s und t besitzen die Länge m beziehungsweise n. Unter der Randbedingung, dass die Reihenfolge der Nachrichten erhalten bleibt, lässt sich aus s und t eine Menge zwischen einer und höchstens $\binom{m+n}{n}$ verschiedenen Nachrichtensequenzen der Länge $m+n$ erzeugen. Genau eine Nachrichtensequenz ergibt sich dann, wenn die Nachrichtensequenzen Folgen von nur einer Nachricht sind oder eine leere Nachrichtensequenz involviert ist. Die maximale Anzahl erhält man, wenn in den Nachrichtensequenzen keine Nachricht mehrfach vorkommt. Betrachten wir zur Illustration das folgende einfache Beispiel zweier Nachrichtensequenzen mit drei beziehungsweise zwei verschiedenen Nachrichten.

Beispiel (Überlappung von Nachrichtensequenzen)
Seien s und t Nachrichtensequenzen mit $s = \langle m_1, m_2, m_3 \rangle$ und $t = \langle m_4, m_5 \rangle$. $s \oplus t$ ist dann die Menge der folgenden 10 Nachrichtensequenzen:

$$\begin{aligned} s \oplus t = \{ \quad & \langle m_1, m_2, m_3, m_4, m_5 \rangle, \langle m_1, m_2, m_4, m_3, m_5 \rangle, \langle m_1, m_2, m_4, m_5, m_3 \rangle, \\ & \langle m_1, m_4, m_2, m_3, m_5 \rangle, \langle m_1, m_4, m_2, m_5, m_3 \rangle, \langle m_1, m_4, m_5, m_2, m_3 \rangle, \\ & \langle m_4, m_1, m_2, m_3, m_5 \rangle, \langle m_4, m_1, m_2, m_5, m_3 \rangle, \langle m_4, m_1, m_5, m_2, m_3 \rangle, \\ & \langle m_4, m_5, m_1, m_2, m_3 \rangle \}. \end{aligned}$$

Die Reihenfolge der Nachrichten der einzelnen Nachrichtensequenzen bleibt erhalten, zum Beispiel ist in den Nachrichtensequenzen von $s \oplus t$ stets die Nachricht m_4 der Nachrichtensequenz t vor der Nachricht m_5 zu finden. □

Gezeitete Nachrichtenströme sind, wie oben beschrieben, unendliche Sequenzen von endlichen Nachrichtensequenzen, wobei jede Nachrichtensequenz genau einem Zeitintervall zugeordnet ist. Seien nun mit $i \in \mathbb{N}_+$ die Zeitfenster jeweils identifiziert, dann lässt sich die Definition der Überlappungsmenge von Nachrichtensequenzen leicht auf Nachrichtenströme erweitern.

Definition 3.9 (Überlappung von Nachrichten in Strömen)

Die Überlappungsmenge zweier Nachrichtenströme erhält man, indem für jedes Zeitintervall der Überlappungsoperator auf die dort enthaltenen Nachrichtensequenzen angewandt wird.
Seien s und t zwei Nachrichtenströme. $s \oplus t$ ist dann die Menge folgender Nachrichtenströme:

$$s \oplus t \stackrel{\mathsf{def}}{=} \{ r : i \in \mathbb{N}_+ \ \wedge \ r.i = s.i \oplus t.i \}$$

□

Da eine Kanalhistorie aus Nachrichtenströmen für eine Menge von Kanälen besteht, kann nun auf der Grundlage der Überlappungsmenge von Nachrichtenströmen auch die

Überlappungsmenge von Kanalhistorien definiert werden. Es sei daran erinnert, dass mit $x.c$ bei einer Kanalhistorie x auf den Nachrichtenstrom des Kanals c referenziert wird.

Definition 3.10 (Überlappung von Nachrichten in Kanalhistorien)

Die Überlappungsmenge zweier Kanalhistorien erhält man, indem für jeden gemeinsamen Kanal der Überlappungsoperator auf die Nachrichtenströme dieser Kanäle angewandt wird.
Seien $x \in \overrightarrow{C}$ und $y \in \overrightarrow{D}$ zwei Kanalhistorien. Dann ist $(x \oplus y) \subseteq \overrightarrow{C \cup D}$ die Menge der möglichen überlappenden Kanalhistorien von x und y. Für alle Nachrichtenströme von $x \oplus y$ gilt:

$$(x \oplus y).c \stackrel{\text{def}}{=} \begin{cases} x.c \oplus y.c & \text{falls } c \in \mathsf{cid}(C) \cap \mathsf{cid}(D) \\ x.c & \text{falls } c \in \mathsf{cid}(C) \backslash \mathsf{cid}(D) \\ y.c & \text{falls } c \in \mathsf{cid}(D) \backslash \mathsf{cid}(C) \end{cases}$$

In jeder Historie dieser Menge sind alle Nachrichten der Nachrichtenströme von x und y enthalten, und zwar sowohl in derselben Reihenfolge, als auch in denselben Zeitfenstern.

□

Zeitliche Verschiebung von Kanalhistorien

Der Überlappungsoperator liefert eine Menge von Kanalhistorien, die sich aus der Überlappung der Nachrichten erzeugen lassen, welche jeweils in demselben Zeitfenster vorkommen. Oftmals ist man allerdings auch an den Auswirkungen interessiert, wenn sich die Nachrichten einer Kanalhistorie verzögern. Wir führen dafür einen Operator ein, mit dessen Hilfe die Nachrichten einer Kanalhistorie um eine bestimmte Anzahl von Zeiteinheiten verschoben werden können.

Definition 3.11 (Verzögerte Nachrichten in Kanalhistorien)

Sei $x \in \overrightarrow{C}$ eine Kanalhistorie. Der Operator

$$\mathsf{delay} \in \overrightarrow{C} \times \mathbb{N} \times \mathbb{N} \mapsto \overrightarrow{C}$$

liefert eine Kanalhistorie, bei der die Nachrichten auf allen Kanälen gegenüber denen in x ab dem Zeitpunkt n um m Zeiteinheiten verzögert sind.

$$\begin{aligned} \mathsf{delay}(x, n, m) \stackrel{\text{def}}{=} \{ z \in \overrightarrow{C} : \quad & (z.c).i = (x.c).i && \text{falls } i < n \\ \wedge \quad & (z.c).i = (x.c).(i - m) && \text{falls } i \geq n + m \\ \wedge \quad & (z.c).i = \langle\,\rangle && \text{falls } n \leq i < n + m \} \end{aligned}$$

□

Eine Verschiebung von Nachrichten um 0 Zeiteinheiten ergibt stets die ursprüngliche Kanalhistorie. Für alle Kanalhistorien x und alle $n \in \mathbb{N}$ gilt $\mathsf{delay}(x, n, 0) = x$.

3.3 Kombination von Funktionen

Wie in Abschnitt 3.1 gezeigt, wird die syntaktische Schnittstelle einer Komponente oder einer Nutzungsfunktion mit einer Menge von Eingabekanälen und einer Menge von Ausgabekanälen festgelegt. Das Verhalten – oder auch das *Schnittstellenverhalten* – wird nun durch eine Funktion f modelliert, und zwar durch die Abbildung der Menge der Eingabehistorien auf die Menge der Ausgabehistorien,

$$f : \overrightarrow{I} \mapsto \wp(\overrightarrow{O})$$

Mit der Potenzmenge von $\overrightarrow{O}$ wird potenziellem nicht deterministischem Verhalten Rechnung getragen. Für $x \in \overrightarrow{I}$ wird mit $f.x$ auch die Menge der (nicht deterministischen) Ausgabehistorien bezeichnet (siehe auch [Bro03a, Bro05]),

$$f.x \stackrel{\mathsf{def}}{=} \{y : x \in \overrightarrow{I} \wedge y \in f(x)\}$$

Partiell definiertes Verhalten

In der Einführung wurde bereits der Begriff des Diensts informell als eine Menge von Verhaltensmustern definiert, welcher die Nutzung eines Systems zu einem bestimmten Zweck charakterisiert. Die FOCUS Theorie wurde in [Bro03a, Bro05] nun um den Dienstbegriff erweitert. Ein Dienst wird dort formal analog der Komponentendefinition in [BS01] spezifiziert, denn er besitzt eine syntaktische Schnittstelle und sein Verhalten erfüllt das Kausalitätsprinzip. Die Ausgaben sind also ausschließlich durch vorherige Eingaben bestimmt. Der Unterschied liegt allerdings darin, dass ein Dienst nicht total spezifiziert sein muss, wie dies bei einer Komponente der Fall ist.

Auch wir werden für die syntaktische und semantische Beschreibung eines Diensts beziehungsweise einer Nutzungsfunktion die oben eingeführte Notation verwenden. Das Schnittstellenverhalten wird ebenso als eine Relation von Eingabehistorien und Ausgabehistorien modelliert. Das Schnittstellenverhalten wird mit der Definition partiellen Verhaltens und der Spezifikation einer Domäne verallgemeinert.

Domäne und Wertebereich einer Nutzungsfunktion

Eine Nutzungsfunktion wird typischerweise partiell definiert, das heißt, das Verhalten bei beliebigen Eingaben ist von untergeordnetem Interesse. Nur das Verhalten bei eingehaltenen Konventionen ist zu spezifizieren. Es ist also möglich, dass eine Nutzungsfunktion

f für bestimmte Eingabehistorien $x \in \overrightarrow{I_f}$ keine Ausgabehistorien definiert, also $f.x = \varnothing$. Im Folgenden benötigen wir daher den Begriff der Domäne (dom) einer Funktion.

Definition 3.12 (Domäne einer Funktion)

Sei f eine Funktion mit der syntaktischen Schnittstelle $(I_f \blacktriangleright O_f)$. Die Menge aller gültigen Eingabehistorien

$$\mathsf{dom}(f) = \{x \in \overrightarrow{I_f} : f.x \neq \varnothing\}$$

wird als Domäne von f bezeichnet. $\mathsf{dom}(f)$ enthält alle Eingabehistorien, welche die Benutzungskonventionen für f erfüllen. □

Aus der Domäne ergibt sich der Wertebereich (ran) einer Funktion.

Definition 3.13 (Wertebereich einer Funktion)

Sei f eine Funktion mit der syntaktischen Schnittstelle $(I_f \blacktriangleright O_f)$. Die Menge der Ausgabehistorien von f für alle gültigen Eingaben

$$\mathsf{ran}(f) = \{y : y \in f.x \wedge x \in \mathsf{dom}(f)\}$$

wird als Wertebereich (engl. „range“) von f bezeichnet. □

Falls für eine Funktion f für alle Eingabehistorien x keine Ausgabehistorie definiert ist, $\forall\, x \in \overrightarrow{I_f} : f.x = \varnothing$, wird diese als *paradoxe Funktion* bezeichnet. Es gilt $\mathsf{dom}(f) = \varnothing$.

Teilverhalten

Bei einer Kanalhistorie interessiert oft nur ein Teil der Nachrichtenströme. Gerade bei kombinierten Funktionen möchte man nur die Nachrichten auf einer bestimmten Schnittstelle beobachten. Präziser formuliert: man ist an den Nachrichtenströmen nur für eine Teilmenge der Kanäle mit den in deren Trägermengen definierten Nachrichten interessiert.

Definition 3.14 (Projektion einer Historie auf getypte Kanäle)

Seien C und D zwei Mengen getypter Kanäle, wobei C ein Subtyp von D ist, $C \subseteq D$. Für eine Kanalhistorie $x \in \overrightarrow{D}$ ist die auf die Kanalmenge C projizierte Kanalhistorie $x \,|\, C$ folgendermaßen definiert:

$$x \,|\, C \stackrel{\mathsf{def}}{=} \{x' : x'.c = \mathsf{car}(T) \,Ⓢ\, x.c \ \wedge\ (c\colon T) \in C\}$$

Wir verwenden diese Projektionsschreibweise im Folgenden auch abkürzend bei einer Menge von Kanalhistorien. Für eine Menge von Kanalhistorien $X \subseteq \overrightarrow{D}$ ist der Projektionsoperator dabei auf jede Historie der Menge anzuwenden.

$$X \,|\, C \stackrel{\text{def}}{=} \{x \,|\, C : x \in X\}$$

□

Um Verhaltensabhängigkeiten bei kombinierten Funktionen bewerten zu können, muss das zum Gesamtverhalten der Kombination beitragende Teilverhalten einer Funktion ausgedrückt werden können. Dies erreicht man durch Einschränkung des Verhaltens auf einen Ausschnitt der syntaktischen Schnittstelle, was also einer Projektion auf Teilmengen der Eingabe- und Ausgabekanäle gleichkommt. Dafür werden auch die Begriffe „Splicing“ [Bro05] oder „Restriction“ [SS03b] verwendet.

Seien die syntaktischen Schnittstellen der Funktionen f und s in einer Subtyprelation, $(I_f \blacktriangleright O_f) \subseteq_{\text{IF}} (I_s \blacktriangleright O_s)$. Um nun auf das auf die Schnittstelle $(I_f \blacktriangleright O_f)$ eingeschränkte Verhalten von s zu referenzieren, wird als Schreibweise

$$s\dagger(I_f \blacktriangleright O_f)$$

verwendet. Im Wesentlichen entspricht dies einer Filterfunktion für Kanäle und Nachrichtentypen, welche die Eingabehistorien auf I_f und die Ausgabehistorien auf O_f projiziert. Mit $s\dagger(I_f \blacktriangleright O_f).x$ werden ausschließlich Eingaben für s auf den Kanälen I_f betrachtet, also $x \in \overrightarrow{I_f}$, und Ausgaben auf anderen Kanälen als aus O_f werden ignoriert. Für alle $x \in \overrightarrow{I_f}$ gilt

$$s\dagger(I_f \blacktriangleright O_f).x = \{y \,|\, O_f : \exists\, x_s \in \overrightarrow{I_s} : x = (x_s \,|\, I_f) \ \wedge \ y \in s.x_s\}$$

Verhaltensverfeinerung

Der Begriff der Verhaltensverfeinerung taucht insbesondere im Zusammenhang mit der schrittweisen Entwicklung von Anforderungsspezifikationen im Rahmen des Requirements Engineerings auf [BS01]. Durch das Hinzufügen von mehr Information durch weitere oder präzisere funktionale Anforderungen wird die Unterspezifikation von Funktionen sukzessive reduziert. Angenommen zwei Funktionen f und s besitzen dieselbe syntaktische Schnittstelle $(I \blacktriangleright O)$. Wenn der Grad der Unterspezifikation einer Funktion f höher ist als bei einer Funktion s, das Verhalten von s also einer Verfeinerung von f entspricht, so wird dies durch

$$\forall\, x \in \overrightarrow{I} : s.x \subseteq f.x$$

dargestellt. Falls die Schnittstelle von s durch Hinzunahme von Kanälen und Nachrichtentypen umfangreicher ist, $(I_f \blacktriangleright O_f) \subseteq_{\text{IF}} (I_s \blacktriangleright O_s)$, kann man auch nur das auf bestimmte Eingabe- und Ausgabekanäle eingeschränkte Verhalten in Betracht ziehen.

Definition 3.15 (Verhaltensverfeinerung)

Gegeben seien zwei Funktionen f und s und ihre syntaktischen Schnittstellen stehen in der Subtyprelation $(I_f \blacktriangleright O_f) \subseteq_{IF} (I_s \blacktriangleright O_s)$. Die Funktion s ist eine Verhaltensverfeinerung der Funktion f genau dann, wenn gilt:

$$\forall\, x \in \overrightarrow{I_f} : s\dagger(I_f \blacktriangleright O_f).x \subseteq f.x$$

□

Schnittstellenverhalten kombinierter Funktionen

Bei dem Versuch, Funktionen zu kombinieren, werden die isolierten Spezifikationen einzelner Funktionen von einer gemeinsamen Betrachtungsweise abgelöst. Das Wort „Versuch“ wird an dieser Stelle bewusst verwendet, denn die Bedeutung der Kombinierbarkeit der Funktionen wird erst später geklärt. Selbstverständlich sollte letztendlich nur von einer „Kombination von Funktionen“ gesprochen werden, sofern die Funktionen auch kombinierbar sind. Die notwendigen Eigenschaften kombinierbarer Funktionen sollen hier jedoch noch nicht Gegenstand der Untersuchung sein, sondern werden ausführlich in Kapitel 4 diskutiert.

Die Kombination von Funktionen entspricht aus syntaktischer Perspektive formal der Vereinigung der Eingabekanäle und der Vereinigung der Ausgabekanäle der Funktionen – gemäß den oben definierten Mengenoperationen auf Kanalmengen. Für eine Funktion f mit der syntaktischen Schnittstelle $(I_f \blacktriangleright O_f)$ und eine Funktion g mit der syntaktischen Schnittstelle $(I_g \blacktriangleright O_g)$ wird der Kombination der Funktionen die syntaktische Schnittstelle $(I_f \cup I_g \blacktriangleright O_f \cup O_g)$ zugrunde gelegt.

Die Schnittstelle einer potenziellen Superfunktion s ist dann ein Supertyp der kombinierten Schnittstelle:

$$(I_f \cup I_g \blacktriangleright O_f \cup O_g) \subseteq_{\text{IF}} (I_s \blacktriangleright O_s)$$

Wie wir im Folgenden sehen werden, ist die syntaktische Voraussetzung zwar notwendig, jedoch nicht hinreichend für die Kombinierbarkeit von Funktionen. Nur für den einfachen Fall, dass es keine Seiteneffekte gibt, entspricht das Schnittstellenverhalten der in eine Superfunktion eingebettete Funktion dem isolierten Teilverhalten der Funktion. Für eine Funktion f in s ist dann etwa folgender Zusammenhang gegeben:

$$\forall\, x \in \overrightarrow{I_s} : (s.x) \,|\, O_f = f.(x \,|\, I_f)$$

In der nachstehenden Übersicht sind nochmals die formalen Schreibweisen zusammengefasst, welche in folgenden Kapiteln für die Beschreibungen von Funktionszusammenhängen gebraucht werden. Die Begriffe „Eingaben“ und „Ausgaben“ stehen hier abkürzend für eine Menge von Eingabehistorien beziehungsweise eine Menge von Ausgabehistorien.

	Bedeutung
$\mathsf{dom}(s)$, $\mathsf{dom}(g)$	Die Menge aller gültigen Eingaben eines Supersystems s beziehungsweise einer Funktion g.
$x \in \mathsf{dom}(s)$	Eine gültige Eingabe x eines Supersystems s.
$x' \in \mathsf{dom}(g)$	Eine gültige Eingabe x' einer betrachteten Funktion g.
$s.x$	Die Menge der Ausgaben eines Supersystems s für eine Eingabe x. Wenn x nicht aus der Domäne von s ist, gibt es keine definierten Ausgaben, also $s.x = \varnothing$.
$g.x'$	Die Menge der Ausgaben einer betrachteten Funktion g für eine Eingabe x'. Diese Menge entspricht im Allgemeinen auch den erwarteten beziehungsweise vorgabegemäßen Ausgaben, falls g in ein Supersystem s eingebettet ist. Wenn x' nicht aus der Domäne von g ist, gibt es keine definierten Ausgaben, also $g.x' = \varnothing$.
$x' = x \mid I_g$	Eingabe x eines Supersystems eingeschränkt auf die Eingabeschnittstelle I_g von g.
$\{x \in \mathsf{dom}(s) : (x \mid I_g) \in \mathsf{dom}(g)\}$	Alle gültigen Eingaben eines Supersystems s, die eingeschränkt auf I_g auch gültige Eingaben von g sind.
$s\dagger(I_g \blacktriangleright O_g).x'$	Ein Supersystem s erhält nur eine Eingabe x' auf der eingeschränkten Schnittstelle I_g. Alle anderen potenziellen und möglicherweise benötigten Eingaben auf anderen Kanälen werden ignoriert. Auch mögliche Ausgaben auf anderen Kanälen als solche in O_g werden ignoriert. Beide Einschränkungen können dazu führen, dass die zu beobachtende Ausgabemenge leere Historien enthalten kann. Mit diesem Term wird die Ausgabe einer in s eingebetteten Funktion g für eine Eingabe x' beschrieben. Mögliche Seiteneffekte, verursacht über Eingaben auf anderen Eingabekanälen, werden hierbei ignoriert.

	Bedeutung
$(s.x) \mid O_g$ bzw. $\{y \mid O_g : y \in s.x\}$	Die Menge der auf die Schnittstelle O_g eingeschränkten Ausgaben eines Supersystems s für eine Eingabe x. Mögliche Ausgaben auf anderen Kanälen als solche in O_g werden ignoriert. Diese Einschränkung kann dazu führen, dass sich hier die zu beobachtende Ausgabemenge reduziert und auch leere Historien enthalten kann.
$x' \in \mathsf{dom}(s \dagger (I_g \blacktriangleright O_g))$	Eine gültige Eingabe x' an der eingeschränkten Schnittstelle $(I_g \blacktriangleright O_g)$ eines Supersystems s. Die Restriktion der Eingabe auf I_g kann die Menge der möglichen Ausgaben erhöhen (Nichtdeterminismus), mit der Restriktion auf O_g wird die Menge der Ausgaben jedoch möglicherweise reduziert. Die Menge der Ausgaben ist aber auch auf der eingeschränkten Schnittstelle nicht leer.

3.4 Notationen

Für die funktionale Beschreibung von Funktionen gibt es je nach Modellierungsumgebung, Detaillierungsgrad oder auch Präferenz eine große Anzahl verschiedener Techniken (vergleiche Kapitel 2). Am zweckdienlichsten erscheint im Rahmen dieser Arbeit die Verwendung entweder von Message Sequence Charts [OMG04, Krü04] oder von Zustandsautomaten, wobei bei letzteren insbesondere die Eingabehistorien und Ausgabehistorien der Funktionen deutlicher zum Vorschein kommen.

In dieser Arbeit werden daher die Funktionen als – in der Regel partielle – Zustandsautomaten mit Eingabe und Ausgabe spezifiziert. Zur Beschreibung dieser Zustandsautomaten verwenden wir so genannte Zustandsübergangsdiagramme [BS01]. Eine Eingabenachricht ist dort durch ein Fragezeichen (*?*) und eine Ausgabenachricht durch ein Ausrufezeichen (*!*) zwischen Kanalbezeichnung und Nachricht gekennzeichnet. Im Zusammenhang mit der Spezifikation von Funktionen eines Mobiltelefons in Kapitel 6 werden etwa die Kanäle **kbd** (Tasten), **dsp** (Anzeige) und **sys** verwendet.

Wie in Abbildung 3.2 gezeigt, sind Kontrollzustände einer Funktion durch Ellipsen dargestellt, während Datenzustände gegebenenfalls als Vor- und Nachbedingungen bei den Transitionen beschrieben werden. Jede Transition ist also gekennzeichnet durch eine optionale Vorbedingung in geschweiften Klammern *{...}*, eine Eingabenachricht eines Kanals, eine Ausgabenachricht eines Kanals und eine optionale Nachbedingung, ebenfalls in geschweiften Klammern.

Alle derart beschriebenen Kontroll- und Datenzustände werden zunächst nicht in system-

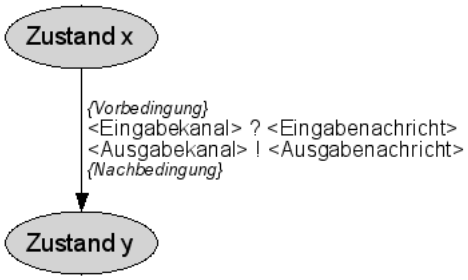


Abbildung 3.2: Transition eines Zustandsautomaten mit Eingabe und Ausgabe

weite oder funktionslokale Zustände klassifiziert. Prinzipiell besitzen daher alle Funktionen einen gemeinsamen Zustandsraum. Die Identifizierung funktionslokaler Zustände ist aus Nutzungssicht im Zuge eines Requirements Engineering nicht relevant und kommt erst in späteren Entwicklungsphasen zum Tragen. Gemeinsame übergreifende Zustände von Funktionen allerdings spiegeln sich als nicht-triviale Zusammenhänge zwischen den Funktionen wider. Solche funktionsübergreifenden Zustände gilt es mithilfe von festgelegten Funktionsbeziehungen zu erkennen.

Die Funktionsdefinitionen mit Zustandsübergangsdiagrammen sind üblicherweise hochgradig partiell. Teilweise ist in den Beispielen der Fallstudie aus Kapitel 6 in einem Zustand für nur genau eine Nachricht[2] über den Eingabekanal **kbd** ein Nachfolgezustand definiert. Anhand eines so beschriebenen Zustandsübergangsdiagramms lässt sich im Übrigen leicht die Domäne einer Funktion als Menge von Eingabenachrichtensequenzen ableiten, indem lediglich die Nachrichten der Eingabekanäle verfolgt werden.

Beziehungstyp und Funktionsbeziehung

Um die Kombinierbarkeit von Funktionen bewerten zu können, sind deren Zusammenhänge und Abhängigkeiten zu analysieren. Ein *Beziehungstyp* umfasst nun allgemein die Eigenschaften, die eine bestimmte Art des Zusammenhangs von Funktionen charakterisieren. Eine konkret formulierte *Funktionsbeziehung* dagegen ist von einem bestimmten Beziehungstyp und legt darüber hinaus die Richtung der Beziehung fest. Für die Bezeichnungen der Beziehungstypen werden in dieser Arbeit englische Begriffe verwendet. Damit sollen einerseits mögliche Verwechslungen zwischen Beziehungstyp und dessen Beschreibung vermieden werden. Andererseits sollen damit aber auch griffige Namen zur Verfügung stehen, welche ein intuitives Verständnis nahe legen.

Zur Formulierung einer Funktionsbeziehung wird die Prädikatenschreibweise oder die

[2] Präziser ausgedrückt: ein Element aus der Trägermenge der Eingabenachrichten.

Infixschreibweise, sowie die graphische Notation gleichberechtigt verwendet. Die Bezeichnung des Beziehungstyps wird dabei gewissermaßen als Operator verwendet. Die Richtung der Beziehung wird dabei von „links nach rechts“ unterstellt.

Beispiel (Schreibweisen für eine Funktionsbeziehung)

Wir illustrieren die genannten Schreibweisen anhand des Beziehungstyps enable (vergleiche Kapitel 5.2) und den Funktionen f und g.

Prädikatenschreibweise

enable(f, g)

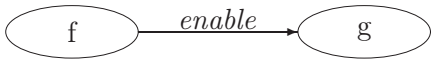
Gemäß der festgelegten Lesekonvention wird also hier f die Funktion g ermöglichen.

Infixschreibweise

‘f’ **enables** ‘g’

Diese Schreibweise ist meist besser lesbar, da vor allem die Richtung der Beziehung intuitiv klar ist. Wenn es passend ist, wird zusätzlich der Buchstabe **s** an die Typbezeichnung angehängt, um die Lesbarkeit zu steigern. Damit wird also nicht etwa auf einen anderen Beziehungstyp referenziert.

Graphische Notation



□

Kapitel 4

Grundlegende Funktionsbeziehungen

Wie in der Einführung erläutert, gehören zu einem geeigneten Produktmodell verschiedene Perspektiven, welche neben der logischen Komponentensicht und der technischen Realisierungssicht insbesondere auch eine Nutzungssicht (siehe Abbildung 1.3) umfassen. Die Nutzungssicht entspricht jedoch klassischerweise der Blackboxsicht auf ein System, bei der im Grunde nur die Eingaben und die Ausgaben des Systems beobachtet werden. In dieser Arbeit wird die Blackboxsicht nun differenzierter betrachtet, um schließlich zu einer Nutzungsarchitektur zu gelangen und somit der Blackbox ebenfalls eine Struktur aufzuprägen. Ausschlaggebend hierfür sind Beziehungen zwischen den Nutzungsfunktionen eines Systems.

Bevor wir im Detail Beziehungstypen untersuchen, beschäftigen wir uns zunächst mit verschiedenen Klassen von Beziehungen. Unter anderem werden Funktionsbeziehungen vorgestellt, mit denen Funktionshierarchien gebildet werden können. Im weiteren Verlauf dieses Kapitels beschreiben wir solche Beziehungen mit einer Subfunktionsrelation beziehungsweise mit einer Superfunktionsrelation. Wir führen dazu den subfunction Beziehungstyp formal ein.

Danach werden die zentralen Begriffe der Kombinierbarkeit und der Unabhängigkeit von Funktionen diskutiert. Um Beziehungen oder Abhängigkeiten zwischen Funktionen zu identifizieren, die über Interaktionsschnittstellen hinaus gehen, ist eine gesamtheitliche Betrachtung der Funktionen samt eines umfassenden Supersystems notwendig. Je nach betrachteter Subfunktion lassen sich die Eingabehistorien dieses Supersystems gemäß des Verhaltens klassifizieren und zur Beschreibung von Beziehungstypen heranziehen.

4.1 Die Nutzungssicht auf ein System

Das Gesamtverhalten eines Systems ist stets das Resultat der Kombination aller Funktionalitäten. Gerade aus der Nutzungsperspektive reicht daher für die Gewinnung eines Systemverständnisses die Identifizierung der Funktionalitäten eines Systems alleine nicht aus. Darüber hinaus ist die klare Herausstellung der Zusammenhänge mit einer expliziten Beschreibung der Beziehungen zwischen den Nutzungsfunktionen essenziell.

Was jedoch zur Beschreibung des Verhaltens logischer Komponenten ausreichend erscheint, genügt nicht für die Charakterisierung funktionaler Zusammenhänge aus der Nutzungssicht. Die in Kapitel 2 untersuchten Ansätze bieten dafür keine oder nur sehr rudimentäre Möglichkeiten an. In der UML sind zum Beispiel die Konzepte `«include»` und `«extend»` für die Modellierung von Nutzungsfällen mit Use Case Diagrammen verankert. Im Rahmen eines Systementwurfs mögen diese Stereotypen ausreichend sein. Sie helfen jedoch für die Entwicklung eines Systemverständnisses nur sehr begrenzt. Das Verhalten eines Systems, dessen Zusammenhänge rein mit diesen Mitteln beschriebenen sind, wird nur in den seltensten Fällen nachvollziehbar sein. In [Mar03] schlägt Robert C. Martin sogar vor, Beziehungen dieser Art für die Modellierung erst gar nicht in Betracht zu ziehen:

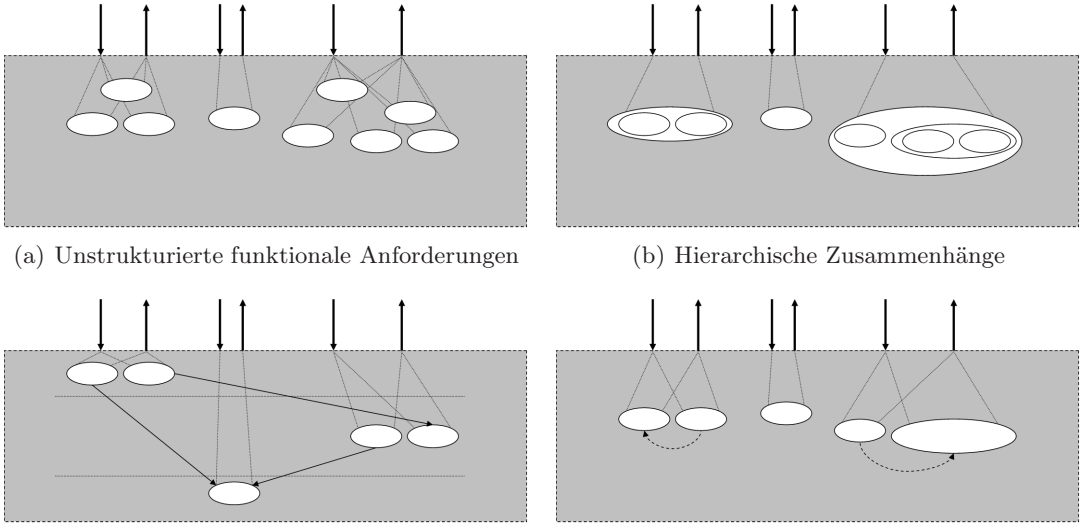
> „I suggest that you actively ignore them. They'll add no value to your use cases, or to your understanding of the system, and they will be the source of many never ending debates about whether or not to use `«extends»` or `«generalization»`."

Auf der anderen Seite ist die pauschale Ignorierung von Beziehungen zwischen Funktionen gerade bei der Entwicklung multifunktionaler Softwaresysteme riskant, nur weil keine geeigneten Beschreibungsmittel zur Verfügung stehen oder um einer unnötigen Diskussion über Modellierungsvorlieben auszuweichen. Wie wir im Folgenden sehen, gibt es je nach Betrachtungsweise vielfältigste Arten von Beziehungen mit unterschiedlichen Konsequenzen für die Kombination von Funktionen.

Perspektiven einer Nutzungsarchitektur

Den Ausgangspunkt für die Entwicklung einer Nutzungsarchitektur bildet eine Menge von geforderten Funktionen, welche als funktionale Anforderungen an das System formuliert sind. Solange keine Beziehungen zwischen diesen Nutzungsfunktionen angenommen werden, sind alle als gleichwertig im Sinne einer uneingeschränkten Kombinierbarkeit zu betrachten. Diese Situation ist in Abbildung 4.1(a) als unstrukturierte Sammlung von Funktionen, welche als Ellipsen gezeichnet sind, dargestellt.

Nun lässt sich aber diese Blackboxsicht mithilfe verschiedener Perspektiven strukturieren, um auch die Zusammenhänge der Nutzungsfunktionen transparent und insbesondere



(a) Unstrukturierte funktionale Anforderungen

(b) Hierarchische Zusammenhänge

(c) Nutzungsabhängigkeiten als Schichtenmodell

(d) Verhaltensabhängigkeiten

Abbildung 4.1: Perspektiven einer Nutzungsarchitektur

nachvollziehbar zu machen. Potenzielle Beziehungen lassen sich etwa aus dem Blickwinkel einer hierarchischen Strukturierung und aus Verhaltenssicht anhand entsprechender Nutzungsszenarien analysieren.

Die angebotenen Funktionen eines Systems sind im Sinne von Detaillierung im Allgemeinen nicht alle auf demselben Niveau. Manche Funktionen verallgemeinern etwa andere Funktionen oder sie bilden Klammern, welche eine Menge von Funktionen zusammenfassen. Gemäß den Zusammenhängen „beinhaltet“ beziehungsweise „bietet Zugang zu“ lassen sich die Funktionen hierarchisch anordnen. In der Grafik in Abbildung 4.1(b) ist dies durch Ellipsen dargestellt, die wiederum andere Ellipsen beinhalten.

Eine andere Anordnung der Funktionen ergibt sich aus den Nutzungsbeziehungen der Funktionen. Funktionen, welche andere Funktionen nutzen, befinden sich auf höherem Niveau. Diese Art der Perspektive entspricht einer Sicht im Sinne eines Schichtenmodells. In Abbildung 4.1(c) sind die Schichten mit horizontalen Linien abgebildet. Die auf verschiedenen Höhen befindlichen Ellipsen deuten die Schichtzugehörigkeit einzelner Funktionen an. Die Pfeile stellen dabei eine Nutzungsbeziehung dar.

Mit den oben beschriebenen Perspektiven lassen sich nur sehr spezielle Aspekte des Einflusses von Funktionen auf andere Funktionen erfassen. Oft besteht aber ein Zusammenhang zwischen Funktionen, der darüber hinausgeht. Eine Funktion übt Einfluss auf eine andere Funktion aus, insofern als im Vergleich zur spezifizierten Vorgabe abweichendes Verhalten festzustellen ist, zum Beispiel durch Manipulation des Zugangs zu einer

Funktion. Diese Sichtweise, die wir als Verhaltenssicht bezeichnen, umfasst Funktionszusammenhänge, welche insbesondere Aussagen über die Kombinierbarkeit von Funktionen erlauben. In Abbildung 4.1(d) sind diese Zusammenhänge mit gestrichelten Pfeilen angedeutet.

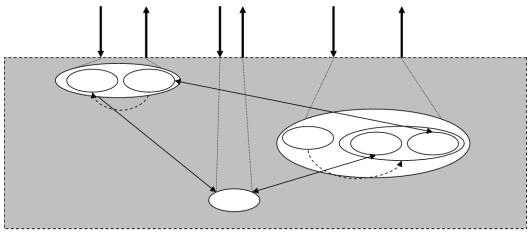


Abbildung 4.2: Strukturierte Blackboxsicht

Durch die Überlagerung der genannten Perspektiven wird die Blackboxsicht eines Systems strukturiert (Abbildung 4.2). Falls es Zusammenhänge zwischen den Funktionen gibt, seien sie hierarchischer oder verhaltenstechnischer Art, so sind sie explizit spezifiziert. Auf diese Weise kann sichergestellt werden, dass ausschließlich kombinierbare Nutzungsfunktionen als Teilfunktionen eines Systems in Betracht gezogen werden.

Beziehungsklassen

Die verschiedenen Blickwinkel der Nutzungssicht bieten sich an, Funktionsbeziehungen zu kategorisieren und sie entsprechend unterschiedlichen Beziehungsklassen zuzuordnen. Die wesentlichen Klassen umfassen daher Klammerbeziehungen (hierarchische Anordnung der Funktionen), Nutzungsbeziehungen und Verhaltensbeziehungen (Querbeziehungen der Funktionen, und zwar losgelöst von hierarchischen Zusammenhängen).

Wie bereits oben erwähnt, liegt das Augenmerk von Klammerbeziehungen auf Enthaltensein- beziehungsweise Teilfunktionsbeziehungen. Derart lassen sich Funktionen organisieren, wie es aus Funktionsnetzen oder Funktionshierarchien bekannt ist [BR01, BR02, WCG04]. Der Zugang zu den Funktionen in der Kombination mit anderen Funktionen, zum Beispiel notwendige Kanalhistorien, steht hier im Blickpunkt des Interesses.

Bei Nutzungsbeziehungen steht traditionell die Bereitstellung von Funktionen, welche auf anderen Funktionen basieren, im Vordergrund. Die Benutzung von Funktionen durch andere Funktionen im Sinne eines Schichtenmodells ist hier maßgeblich. Allerdings verstehen wir in dieser Arbeit unter einer Nutzungsbeziehung nicht ausschließlich den vergleichsweise statischen Zusammenhang, wie er mit „benötigt“ formuliert werden kann. Hier steht eher die dynamische Nutzung einer Funktionalität, sei es mit oder ohne Rückmeldung, im Blickpunkt des Interesses.

Mit Verhaltensbeziehungen werden Zusammenhänge zwischen Funktionen beschrieben, bei denen die Beeinflussung durch möglicherweise nicht spezifikationsgemäßes Verhalten

zu beobachten ist. Zum Beispiel können Nachrichtenflüsse kontrolliert oder Systemzustände geändert werden. Auch die Unabhängigkeit von Funktionen ist eine wesentliche Beziehung. Verhaltensbeziehungen sind typischerweise gerichtete Abhängigkeiten, das heißt: steht eine Funktion in einer bestimmten Beziehung zu einer anderen Funktion, so muss diese Beziehung nicht notwendigerweise auch umgekehrt gelten. Das entscheidende Kriterium ist stets die Kombinierbarkeit der Funktionen – trotz eines möglichen Einflusses einer Funktion auf das Verhalten einer anderen Funktion.

In dieser Arbeit konzentrieren wir uns auf Beziehungstypen der drei genannten Beziehungsklassen und ausschließlich auf potenzielle Zusammenhänge zwischen Funktionen aus Nutzungssicht. Wir vermeiden daher die architekturelle Sichtweise auf ein System mit den Schnittstellen zwischen den logischen Komponenten, um das Vermischen beider Sichten auszuschließen. Insbesondere können im Allgemeinen auch nur aufgrund interner Schnittstellen zwischen Komponenten keine Rückschlüsse auf spezielle Beziehungen zwischen den bereitgestellten Nutzungsfunktionen gezogen werden. Welcher Art eine Beziehung zwischen Nutzungsfunktionen ist, steht nicht in unmittelbarem Zusammenhang wie Nachrichten übertragen werden. Aus Nutzungssicht ist es zunächst sogar irrelevant, ob der Austausch von Nachrichten über interne Kanäle oder auf eine andere Art und Weise erfolgt.

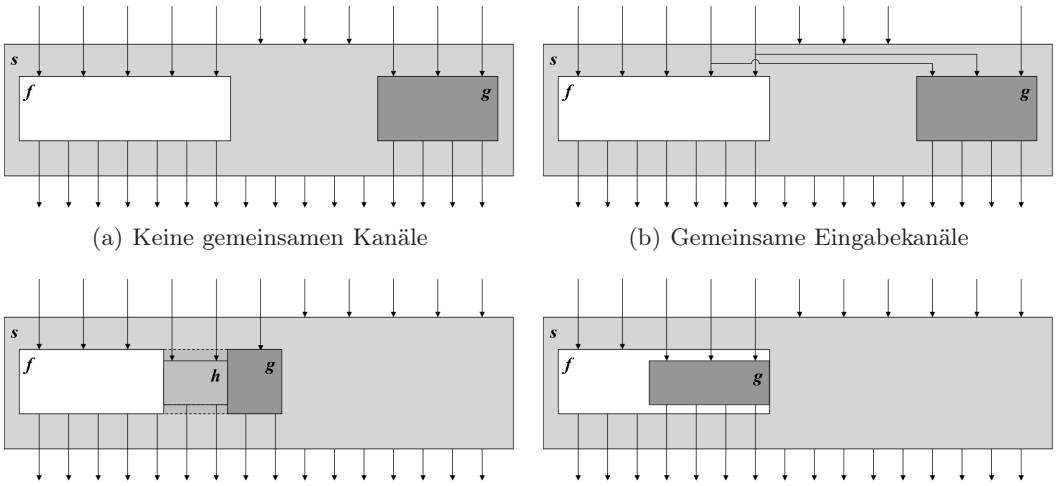
An dieser Stelle sei angemerkt, dass es selbstverständlich noch weitere Beziehungstypen gibt, die den erwähnten Beziehungsklassen nicht zugeordnet werden können. Diese werden hier allerdings nicht ausführlich diskutiert, da sie für die Kombinierbarkeit von Funktionen eine untergeordnete Rolle spielen. Zum Beispiel liegt bei *organisatorischen Beziehungen* das Interesse nicht notwendigerweise auf den funktionalen Zusammenhängen, sondern etwa bei der Abdeckung von Vermarktungsanforderungen. Stichworte wie Funktionsgruppen im Sinne von „Bundles“, Produktvarianten oder „Feature Sets“ sind hier zu erwähnen. Im Rahmen der Feature Oriented Reuse Method (FORM) [KKL+98][KKL+98] werden Funktionen zum Beispiel als *obligatorisch*, *optional* oder *alternativ* gekennzeichnet.

Abschließend seien noch *entwicklungsmethodische Beziehungen* angeführt, die eher dem pragmatischen Entwickeln von Funktionen im Sinne von verifizierbaren Verfeinerungsschritten dienen. Relationen wie zum Beispiel die „Äquivalenz“ oder die „Gleichheit“ von Funktionen werden nicht im Rahmen dieser Arbeit untersucht.

Bevor wir nun beginnen, Funktionsbeziehungen zu diskutieren, stellen wir erst Überlegungen zu möglichen syntaktischen Zusammenhängen an. Seien es gemeinsame oder verschiedene Kanäle, die für die Eingabe oder die Ausgabe benutzt werden. Es sind die ersten Anhaltspunkte zur Beurteilung von abstrakteren Beziehungen.

4.2 Syntaktische Zusammenhänge von Funktionen

Untersuchen wir also zunächst allgemein die möglichen syntaktischen Zusammenhänge zwischen den Schnittstellendefinitionen zweier Funktionen. In Abbildung 4.3 sind die vier relevanten syntaktischen Konstellationen dargestellt, die es zu grundsätzlich zu unterscheiden gilt. Dort ist die Kombination zweier Funktionen f und g derart skizziert, dass sie in ein Supersystem s eingebettet sind.



(a) Keine gemeinsamen Kanäle

(b) Gemeinsame Eingabekanäle

(c) Gemeinsame Eingabe- und Ausgabekanäle

(d) Subtyp einer Schnittstelle

Abbildung 4.3: Syntaktische Zusammenhänge zweier Nutzungsfunktionen

1. Die Funktionen besitzen weder gemeinsame Eingabekanäle noch gemeinsame Ausgabekanäle (Abbildung 4.3(a)).
2. Beide Funktionen besitzen gemeinsame Eingabekanäle, das heißt die Schnittmenge der Eingabekanäle ist nicht leer (Abbildung 4.3(b)).
3. Die Schnittmenge der Eingabekanäle und die Schnittmenge der Ausgabekanäle sind nicht leer (Abbildung 4.3(c)). Hier besteht die Möglichkeit gemeinsamer Teilfunktionen.
4. Die Schnittstelle einer Funktion ist ein Subtyp der Schnittstelle der anderen Funktion (Abbildung 4.3(d)). Hier kann potenziell eine Funktion eine Teilfunktion der anderen Funktion sein.

Die Variante, dass es Ausgabekanäle einer Funktion gibt, die zugleich Eingabekanäle der anderen Funktion sind, wird hier nicht näher betrachtet. Dies sind typischerweise

Beziehungen bei Komponentenarchitekturen, die aus Nutzungssicht nicht relevant sind. In Beschreibungen von logischen Komponentenschnittstellen werden oft die Prädikate „provide“ und „need“ verwendet, um angebotene beziehungsweise benötigte Dienste einer Komponente auszuzeichnen.
Selbstverständlich kann in der Regel durch die (direkte) Übertragung von Nachrichten unmittelbar Einfluss auf das Verhalten einer Funktion genommen werden, insbesondere wenn der Kontrollfluss abhängig von diesen Eingabenachrichten ist. Aus Nutzungssicht lassen sich hier jedoch nur schwer Zusammenhänge herstellen, die über ein Verfügbarkeitsabhängiges Verhalten hinausgehen. Im Kontext der Nutzung von Funktionen durch Funktionen (consult) werden wir später noch kurz darauf eingehen. Doch zurück zu den vier oben genannten syntaktischen Beziehungen.

Im ersten Fall kann es zwar zu keinen Konflikten hinsichtlich des Funktionsverhaltens kommen. Es bedeutet allerdings nicht zugleich, dass eine Funktion die andere nicht beeinflussen kann. Da beide Funktionen potenziell unter dem Dach einer Superfunktion eingebettet sind, kann durch bestimmte Eingaben einer Funktion der Zustand der Superfunktion derart manipuliert werden, dass dies Auswirkungen auf das Verhalten der anderen Funktion hat. Dies ist auch der Grund dafür, warum zur Beurteilung einer Funktionsbeziehung stets das Verhalten eines Supersystems einzubeziehen ist.

Gemeinsame Eingabekanäle von Funktionen bergen grundsätzlich die Chance von gemeinsam definierten Eingabehistorien. Dies muss einerseits nicht notwendigerweise mit der Nicht-Kombinierbarkeit der Funktionen einhergehen. Wenn sich andererseits aber die jeweiligen Ausgaben widersprechen oder auch ersetzt werden, sind – sofern möglich – notwendige Bedingungen für eine konsistente Kombination der Funktionen festzulegen.

Neben den vorher genannten Arten der Beeinflussung ist der vierte Fall, wie bereits erwähnt, eher im Kontext von Subfunktionsrelationen zwischen Funktionen zu sehen. Wenn feststeht, dass eine Funktion die Superfunktion einer anderen Funktion ist, spielen weitere potenzielle Beziehungen zwischen diesen Funktionen nur eine untergeordnete Rolle, da insbesondere die Kombinierbarkeit der Funktionen dann per Definition gegeben ist.

4.3 Hierarchie von Funktionen

Zur Bildung von Funktionshierarchien bietet sich neben der Subfunktionsrelation beziehungsweise der Superfunktionsrelation prinzipiell auch eine Nutzungsrelation an. Die Nutzung von Funktionen durch andere Funktionen basiert auf Zusammenhängen im Sinne eines Schichtenmodells, die im Grunde ebenfalls eine hierarchische Ordnung der Funktionen erlauben. Beide Relationen sind jedoch grundverschieden und dürfen keinesfalls verwechselt werden.

Wir konzentrieren wir uns im Folgenden auf die Subfunktionsrelation. Beispielsweise wird

diese Relation auch im Rahmen der Fallstudie als entsprechendes Funktionsnetz dargestellt. Das Supersystem «Mobiltelefon» besitzt als Subfunktionen etwa die Funktionen «Einschalten», «Ausschalten» oder «Telefonieren». Diese wiederum ist eine Superfunktion von «Anruf annehmen» oder «Anruf beenden», und so fort.

Superfunktion und Teilfunktionen

Betrachten wir allgemein zwei Funktionen f und s, die sowohl gemeinsame Eingabekanäle als auch gemeinsame Ausgabekanäle besitzen, derart dass die syntaktische Schnittstelle von f ein Subtyp der Schnittstelle von s ist.

$$(I_f \blacktriangleright O_f) \subseteq_{\mathrm{IF}} (I_s \blacktriangleright O_s)$$

s ist eine Superfunktion von f genau dann, wenn in s für jede gültige Eingabe von f keine widersprüchlichen Ausgaben entstehen, sich also – etwa durch weitere Funktionen in s – die Menge der Ausgabehistorien als Reaktion auf eine Eingabehistorie nicht vergrößert. Genauer gesagt, wenn in s für alle Eingabehistorien der Domäne von f an den auf die syntaktische Schnittstelle von f projizierten Kanälen keine zusätzlichen Ausgaben entstehen, so ist f eine Subfunktion von s.

Die Eigenschaften des Beziehungstyps subfunction sind somit festgelegt. Bei der Definition stützen wir uns hier auf die in Kapitel 3.3 eingeführte Verhaltensverfeinerungsrelation (siehe Definition 3.15) für Funktionen ab. Eine übergeordnete Funktion s umfasst die syntaktischen Schnittstellen aller ihrer Teilfunktionen und stellt so gleichsam die Kombination dieser Teilfunktionen dar. Das Verhalten einer Teilfunktion f ergibt sich aus der Projektion auf die entsprechende Schnittstelle $(I_f \blacktriangleright O_f)$.

Definition 4.1 (Der Beziehungstyp subfunction)

Gegeben seien zwei Funktionen s und f mit den syntaktischen Schnittstellen $(I_s \blacktriangleright O_s)$ beziehungsweise $(I_f \blacktriangleright O_f)$. Die syntaktische Schnittstelle von f sei ein Subtyp der syntaktischen Schnittstelle von s, $(I_f \blacktriangleright O_f) \subseteq_{IF} (I_s \blacktriangleright O_s)$.

f ist eine Subfunktion von s genau dann, wenn s eine Verhaltensverfeinerung an der Schnittstelle von f ist und die Domäne von s die Domäne von f enthält:

$$\begin{array}{lll} & \forall\, x \in \mathsf{dom}(f) : s\dagger(I_f \blacktriangleright O_f).x \subseteq f.x & \textit{(i)} \\ \wedge & \mathsf{dom}(f) \subseteq \mathsf{dom}(s\dagger(I_f \blacktriangleright O_f)) & \textit{(ii)} \end{array}$$

Wir schreiben dann $\boldsymbol{subfunction}(s, f)$. □

In der Kombination von f mit anderen Subfunktionen von s kann die Menge der Ausgabehistorien durch Präzisierung oder Reduzierung von Unterbestimmtheit verringert werden. Einerseits verhindert nun Bedingung (ii) in Definition 4.1 die paradoxe Funktion

als Verfeinerung. Andererseits erlaubt sie aber mehr Eingabehistorien, mit denen zusätzliche Zugänge zu einer Funktion möglich sind. Synonym wird s auch als *Superfunktion* von f bezeichnet. Als Superfunktion bietet s den Zugang zu f an.

Man beachte in Bedingung (i), dass die Teilmengenrelation nicht für beliebige Historien an den Eingabekanälen I_f, sondern nur für alle gültigen Eingabehistorien von f gelten muss. Denn gäbe es eine Eingabehistorie x', welche nicht in der Domäne von f ist (also $f.x' = \varnothing$), müsste gemäß (i) auch $s\dagger(I_f \blacktriangleright O_f).x' = \varnothing$ gelten. Nun kann es wegen (ii) aber auch möglich sein, dass es ein x' gibt, welches zwar nicht in der Domäne von f, jedoch in der Domäne von $s\dagger(I_f \blacktriangleright O_f)$ ist. Dies würde dann aber in Widerspruch zur Bedingung (i) stehen.

Mit dem subfunction Beziehungstyp kann ein System durch eine Hierarchie der Funktionen strukturiert werden. Wie wir in Abschnitt 4.5 sehen werden, ist dieser Beziehungstyp neben der Bildung von Funktionsnetzen aber insbesondere als Grundlage für die Bewertung der Kombinierbarkeit von Funktionen geeignet. Im Allgemeinen ist letztlich die Existenz von subfunction Beziehungen ausschlaggebend für die Beurteilung der Kombinierbarkeit von Funktionen. Gibt es eine Superfunktion für zwei Funktionen, so sind sie auch kombinierbar und damit potenzielle Teilfunktionen eines Systems.

Da die Subfunktionsrelation eine partielle Ordnung und demzufolge insbesondere transitiv ist, wird mit dem Nachweis der Kombinierbarkeit von Superfunktionen auch automatisch die Kombinierbarkeit von deren Subfunktionen gezeigt. Mit der Strukturierung des Systems in Subfunktionen wird also nicht nur die Übersichtlichkeit gesteigert. In formaler Hinsicht kann somit auch die Anzahl der Kombinierbarkeitsnachweise deutlich reduziert werden.

4.4 Einbettung von Funktionen in ein Supersystem

Mit dem subfunction Beziehungstyp steht ein probates Beschreibungsmittel zur Verfügung, mit dessen Hilfe der hierarchische oder auch „vertikale“ Zusammenhang zwischen zwei Funktionen dokumentiert werden kann. Damit kann aber und soll auch kein Einfluss einer Superfunktion auf deren Subfunktion, oder umgekehrt, angezeigt werden. Hingegen ist diese Relation bei der Beschreibung des Einflusses einer Subfunktion auf eine andere Subfunktion im Kontext eines gemeinsamen Supersystems unumgänglich (ein „horizontaler“ Zusammenhang). Die Charakterisierung des Einflusses einer Nutzungsfunktion auf eine andere ist nur unter Einbeziehung des Verhaltens eines beide Funktionen umfassenden Supersystems sinnvoll.

4.4.1 Die „Dreierkonstellation“ – ein Supersystem mit eingebetteten Funktionen

Wenn wir zur Beschreibung eines Zusammenhangs zwischen zwei Funktionen nichts weiter als die beiden Funktionen selbst in die Betrachtung einbeziehen, kommt als einziger Weg für die Beeinflussung einer Funktion durch die andere nur der Austausch von Nachrichten zwischen den Funktionen in Frage. Mit anderen Worten, dann muss eine Interaktionsschnittstelle zwischen beiden Funktionen existieren.

Allgemein besteht jedoch die Möglichkeit der Beeinflussung einer Funktion durch eine andere Funktion, indem der Zustand eines Systems, in welches beide Funktionen eingebettet sind, geeignet geändert wird. Das heißt, ein beide Funktionen umfassendes Supersystem ist in die Betrachtung einzubeziehen. Sobald also zwei Funktionen zu kombinieren sind, werden sie im Grunde nicht mehr isoliert betrachtet.

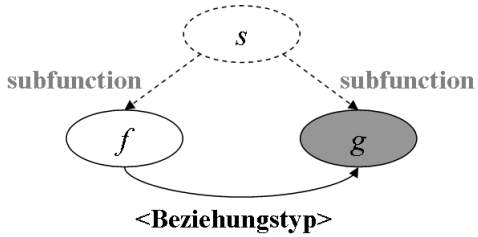


Abbildung 4.4: Beziehungsmuster kombinierbarer Funktionen

Wie in Abbildung 4.4 illustriert, ist daher im Folgenden stets eine „Dreierkonstellation“, bestehend aus

- einer zu betrachtenden Funktion g und
- einer mit g zu kombinierenden Funktion f,
- sowie eines beide Funktionen umfassenden Supersystems s,

der Gegenstand der Untersuchung. Sie bildet die Grundlage dafür, eine Beziehung oder Abhängigkeit zwischen f und g zu identifizieren und zu beschreiben (f und g sind Subfunktionen von s).

Ein Beziehungstyp charakterisiert den Zusammenhang zwischen den in ein Supersystem s eingebetteten Funktionen f und g. Zur Beschreibung einer konkreten Beziehung zwischen den Funktionen müsste neben der Angabe der Funktionen selbst, des Beziehungstyps und der Richtung der Beziehung präziserweise auch stets s mit angeführt werden, beispielsweise

‘f’ **enables** ‘g’ **in** s

Der Einfachheit halber lassen wir in dieser Arbeit den Zusatz „**in** s“ weg, wenn der Bezug zu einem Supersystem implizit gegeben ist.

Änderung des Verhaltens

Wie bereits erwähnt, charakterisiert ein Beziehungstyp einen speziellen Zusammenhang zwischen zwei Funktionen. Eine Funktionsbeziehung geht dabei stets einher mit der Beschreibung des Schnittstellenverhaltens der beobachteten Funktion. Für das von der Vorgabe (isolierte Betrachtungsweise) abweichende Verhalten in Kombination mit anderen Funktionen wird oft auch der Begriff der „Verhaltensänderung“ verwendet.

Man beachte, dass das Verhalten einer Funktion per Spezifikation vorgegeben ist und sich daher auch nicht ändern kann. Wenn im Zusammenhang von Funktionsbeziehungen nun von einer Verhaltensänderung gesprochen wird, bezieht sich dies also nicht auf das Verhalten einer Funktion an sich.

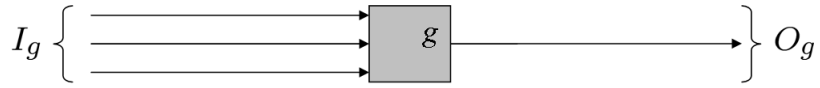
Der Begriff der Verhaltensänderung ist allerdings im Kontext eines Supersystems angebracht, in welches die Funktionen eingebettet sind. Dort kann sich das Verhalten an der Schnittstelle einer betrachteten Funktion sehr wohl im Vergleich zu dem vorgegebenen Ein-/Ausgabeverhalten verändert darstellen. Unterschiedliches Verhalten kann sich insbesondere durch die Berücksichtigung beziehungsweise das Ausblenden von Eingabekanälen ergeben.

Perspektiven zur Verhaltensanalyse

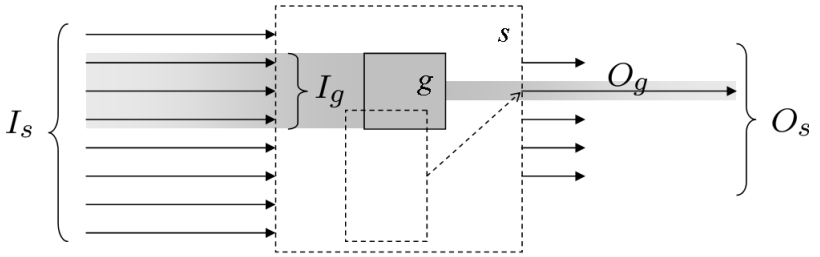
Die Beobachtung des eingebetteten Verhaltens ist demnach ausschlaggebend zur Identifizierung von Beziehungen einer Funktion zu anderen Funktionen. So können zum Beispiel aus der Gegenüberstellung von vorgegebenem Verhalten einer betrachteten Funktion g und Schnittstellenverhalten $s\dagger(I_g \blacktriangleright O_g)$ Rückschlüsse auf den Einfluss anderer Funktionen gezogen werden. Analysiert man den nun den Einfluss auf das Verhalten, sind verschiedene Blickwinkel zu berücksichtigen, die sich speziell auch an den formalen Schreibweisen widerspiegeln.

	Bedeutung
g	Das vorgegebene Verhalten in g.
$s\dagger(I_g \blacktriangleright O_g)$	Das eingebettete Verhalten an der Schnittstelle $(I_g \blacktriangleright O_g)$.
$s\dagger(I_g \cup I_f \blacktriangleright O_g)$	Das Verhalten unter Berücksichtigung einer speziellen eingebetteten Funktion f.
$s \mid O_g$	Das Verhalten unter Berücksichtigung aller in s eingebetteten Funktionen.

Das vorgegebene Verhalten der betrachteten Funktion Bezeichnet g die betrachtete Funktion, so wird mit g zugleich das erwartete beziehungsweise das vorgegebene Schnittstellenverhalten an $(I_g \blacktriangleright O_g)$ spezifiziert.



(a) Isolierte Spezifikation der syntaktischen Schnittstelle



(b) Restriktion der Schnittstelle $s\dagger(I_g \blacktriangleright O_g)$

Abbildung 4.5: Kontext einer betrachteten Funktion g – ein Supersystem s

Das Verhalten an der Schnittstelle der betrachteten Funktion Der Term $s\dagger(I_g \blacktriangleright O_g)$ bezieht sich auf das Schnittstellenverhalten einer Superfunktion s an $(I_g \blacktriangleright O_g)$, in welche g eingebettet ist. Wie in Abbildung 4.5(b) skizziert, sind die Mengen der Eingabekanäle und Ausgabekanäle im Allgemeinen jeweils Obermengen der in den Funktionen spezifizierten Kanäle.

Die grau unterlegten Bereiche deuten hier die Einschränkung auf die in g definierten Kanäle an, um dieses Schnittstellenverhalten von s zu untersuchen. Das Verhalten an $s\dagger(I_g \blacktriangleright O_g)$ kann von den Vorgaben in g prinzipiell aus zweierlei Gründen abweichen.

1. Im Rahmen der Festlegung des Gesamtverhaltens von s werden bei der Integration von Funktionen entsprechende Randbedingungen (Systemzustände) definiert, welche den Zugang zu den Funktionen regeln.

2. Obwohl an der Schreibweise $s\dagger(I_g \blacktriangleright O_g)$ nur schwer erkennbar, kann auch hier bereits der Einfluss einer anderen eingebetteten Funktion f auf das Schnittstellenverhalten wirken, und zwar dann, wenn die Eingabekanäle von f und g nicht disjunkt sind. In Abbildung 4.5(b) wird dies mit einem gestrichelten Rechteck in s dargestellt.

   Gibt es keine gemeinsamen Eingabekanäle, das heißt $(I_f \cap I_g) = \varnothing$, kann ein potenzieller Einfluss von f nur über Eingaben außerhalb von I_g erfolgen. Die Menge der Kanäle, über die Nachrichten eingegeben werden und Einfluss auf das Verhalten haben, ist damit nicht mehr nur auf I_g begrenzt.

Gibt es dagegen eine Funktion f in s mit $(I_f \cap I_g) \neq \varnothing$, die – im Vergleich zu den Vorgaben in g – für unterschiedliches Schnittstellenverhalten verantwortlich ist, muss sie dafür lediglich gemeinsame Nachrichten akzeptieren. Über ihre Ausgaben kann dann der Zustand von s geeignet manipuliert werden.
Die Ausgabekanäle der Funktionen können dabei ohne weiteres disjunkt sein, das heißt $(O_f \cap O_g) = \varnothing$. Es muss also nicht notwendigerweise zu Ausgabekonflikten bei gemeinsam definierten Eingaben kommen. Wenn es darüber hinaus allerdings gemeinsame Ausgabekanäle gibt, ist darauf zu achten, dass die Ausgaben auf diesen Kanälen konsistent sind.

Das Verhalten unter Berücksichtigung aller eingebetteten Funktionen Wir untersuchen hier die Ausgaben auf O_g, die sich unter Berücksichtigung aller sonstigen eingebetteten Funktionen in s mit entsprechenden Eingaben auf gegebenenfalls weiteren Kanälen $(I_g \subseteq I_s)$ ergeben. Als abkürzende Schreibweise wird dafür auch $s \,|\, O_g$ verwendet.

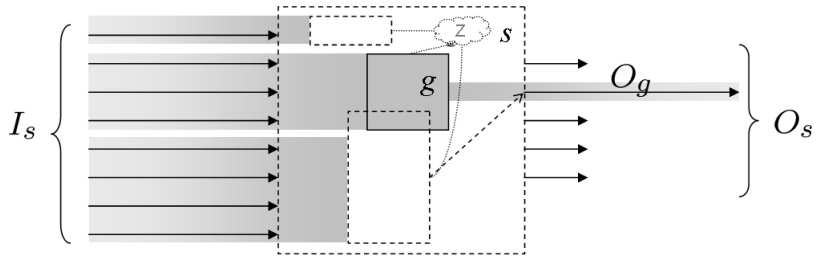


Abbildung 4.6: Restriktion der Ausgabekanäle auf O_g

Im Vergleich zur Darstellung in Abbildung 4.5(b) ist hier in Abbildung 4.6 zusätzlich angedeutet, dass die Eingaben anderer eingebetteter Funktionen, ganz gleich ob sie über gemeinsame oder weitere Kanäle übertragen werden, den Zustand von s manipulieren und damit die Ausgaben beeinflussen können. Unterscheiden sich die Ausgaben für Eingaben auf allen Kanälen I_s nicht von denen, die das Ergebnis der nur auf I_g eingeschränkten Eingaben sind,

$$\forall\, x \in \mathsf{dom}(s) : s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = (s.x) \,|\, O_g$$

so ist dies eine notwendige Voraussetzung dafür, dass g in s „unbeeinflusst eingebettet ist“.

Wie wir sehen werden, ist es durchaus möglich, dass zwar das Verhalten für Eingaben auf I_s und der Restriktion auf I_g identisch ist, aber dennoch nicht der Vorgabe in g entspricht. Ebenso kann nicht ausgeschlossen werden, dass das Schnittstellenverhalten an $(I_g \blacktriangleright O_g)$ zwar konform zu der Vorgabe in g ist, es aber zu unterschiedlichen Ausgaben kommen kann, sobald zusätzliche Eingaben auf anderen Kanälen als I_g berücksichtigt werden.

Das Verhalten unter Berücksichtigung einer speziellen eingebetteten Funktion Die genannten drei Perspektiven mit den entsprechenden Schreibweisen sind essenziell für die nachfolgenden Untersuchungen, um überhaupt potenzielle Abhängigkeiten zwischen einer betrachteten Funktion g und anderen Funktionen zu erkennen. Zur Identifizierung einer konkreten Beziehung zu einer bestimmten Funktion f reichen sie selbstverständlich noch nicht aus. Dafür sind die Auswirkungen auf das Verhalten von g noch speziell unter Berücksichtigung der Schnittstelle von f zu prüfen. Bei der Analyse einer potenziellen Beziehung zwischen f und der betrachteten Funktion g stellt sich nicht nur die Frage, ob das Verhalten von g in Kombination mit f von den Vorgaben abweicht, sondern ob f tatsächlich auch Einfluss auf das Verhalten von g in s hat.

Es gilt also, speziell den Einfluss von f im Kontext von s zu untersuchen. Dabei wird mit $s\dagger(I_g \cup I_f \blacktriangleright O_g)$ auf die Ausgaben an O_g referenziert, und zwar unter Berücksichtigung einer anderen eingebetteten Funktion f in s mit entsprechenden Eingaben auf gegebenenfalls weiteren Kanälen. Die Menge der Eingabekanäle, über die Nachrichten übertragen werden, welche Einfluss auf das Verhalten von g haben, kann dabei durchaus größer als I_g selbst sein.

Es ist allerdings Vorsicht geboten, sich allein auf das unterschiedliche Verhalten für Eingaben auf I_g beziehungsweise auf $I_g \cup I_f$ abzustützen. Selbst wenn für (gegebenenfalls zusätzliche) Eingaben einer Funktion f abweichendes Ausgaben an O_g zu beobachten ist, so ist dies tatsächlich noch kein hinreichendes Kriterium für einen Einfluss von f auf g. Denn

1. müssen lediglich durch gemeinsame Eingaben von f und g nicht zwangsläufig die Ausgaben von g manipuliert werden und
2. können andere eingebettete Funktionen als f, die allerdings gemeinsame Eingaben mit f und g haben, für die Beeinflussung von g verantwortlich sein.

Wir wollen uns dies anhand der folgenden Überlegung veranschaulichen.

> Angenommen man beobachtet nicht vorgabegemäße Ausgaben auf O_g, wenn zusätzliche Eingaben auf weiteren Kanälen erfolgen, insbesondere Eingaben einer Funktion f. Es gibt also ein $x \in \mathsf{dom}(s)$, für das gilt:
>
> $$s\dagger(I_f \cup I_g \blacktriangleright O_g).(x \,|\, I_f \cup I_g) \neq g.(x \,|\, I_g) \ \wedge \ (x \,|\, I_f) \in \mathsf{dom}(f)$$
>
> Daraus unmittelbar zu schließen, dass f die Funktion g beeinflusst, ist nicht richtig, da bereits $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \neq g.(x \,|\, I_g)$ gelten kann. Wenn insbesondere
>
> $$s\dagger(I_f \cup I_g \blacktriangleright O_g).(x \,|\, I_f \cup I_g) = s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$$
>
> gilt, so haben die Eingaben von f offenbar keinen Einfluss auf die Ausgaben von g. Dies ist zum Beispiel gegeben, wenn g erst durch eine andere Funktion als f zu aktivieren ist und es solange zu nicht vorgabegemäßen Ausgaben kommt.

Andererseits kann auch bei

$$(s.x)\,|\,O_g \neq g.(x\,|\,I_g) \;\wedge\; (x\,|\,I_f) \in \mathsf{dom}(f)$$

nicht auf ein Beeinflussung durch f geschlossen werden. Denn auch hier ist es möglich, dass

$$s\!\uparrow\!(I_f \cup I_g \blacktriangleright O_g).(x\,|\,I_f \cup I_g) = s\!\uparrow\!(I_g \blacktriangleright O_g).(x\,|\,I_g)$$

gilt, die Eingaben von f also keinen Einfluss auf die Ausgaben von g haben. Die Manipulation der Ausgabe hat ihre Ursache in Eingaben auf anderen Kanälen anderer eingebetteter Funktionen als f. Beispielsweise ist dies der Fall, wenn g durch eine andere Funktion als f deaktiviert wird.

Zusammenfassend stellen also die genannten Ausdrücke zwar stets notwendige Bedingungen dar, sie sind aber nicht hinreichend für eine tatsächlich existierende Beziehung zwischen den Funktionen.

Fazit Die in den Ausdrücken verwendete Schreibweise zur Einschränkung von Kanälen hat gegebenenfalls auch Konsequenzen für das beobachtete Verhalten hat. Die Restriktion der Ausgabeschnittstelle kann etwa nichtdeterministisches Verhalten aufheben, indem die Menge verschiedener Ausgabehistorien reduziert wird. Eine Änderung des Verhaltens im Sinne einer Manipulation der Ausgaben auf weiteren Kanälen kann mit dieser Restriktion ebenfalls nicht beurteilt werden. Die Restriktion der Eingabeschnittstelle kann dagegen nichtdeterministisches Verhalten zur Konsequenz haben, da so insbesondere von Eingaben abstrahiert wird, welche über Seiteneffekte die Ausgabe beeinflussen können.

Nun kann anhand der – je nach Restriktion der Kanäle – beobachteten Ausgaben der potenzielle Einfluss auf das Verhalten einer Nutzungsfunktion g eingeschätzt werden. Es reicht allerdings nicht aus, nur das Verhalten an der Schnittstelle der betrachteten Funktion $s\!\uparrow\!(I_g \blacktriangleright O_g)$ mit der Verhaltensvorgabe in g zu vergleichen. Um generell die Zusammenhänge zu anderen eingebetteten Funktionen korrekt beurteilen zu können, ist darüber hinaus auch der Vergleich des Verhaltens unter Berücksichtigung aller Eingabekanäle in s anzustellen $(s\,|\,O_g)$.

Zur Identifizierung einer bestimmten verantwortlichen Funktion sind obendrein auch diejenigen Ausgaben gegenüber zu stellen, die aufgrund von Eingaben speziell nur über die Eingabekanäle beider Funktionen entstehen $(s\!\uparrow\!(I_f \cup I_g \blacktriangleright O_g))$. Wie oben gezeigt, ist dabei allerdings zu beachten, dass die Bedingungen im Allgemeinen zwar notwendig, aber keine hinreichenden Kriterien für eine Beeinflussung repräsentieren.

Bevor wir uns im Detail der Diskussion des Verhaltens an den verschiedenen Schnittstellen zuwenden, klassifizieren wir erst die Menge der Eingabehistorien eines Supersystems.

Denn nicht alle Eingaben sind für eine zu betrachtende Funktion relevant oder gar definiert. Wir partitionieren daher die Domäne eines Supersystems s bezüglich des Verhaltens einer eingebetteten, zu betrachtenden Funktion g.

4.4.2 Partitionierung der Domäne eines Supersystems

Die Eingabehistorien der Domäne und die Ausgabehistorien einer Superfunktion s umfassen Nachrichtenströme der Kanäle aller in s eingebetteten Subfunktionen. Je nach betrachteter Subfunktion lassen sich die Eingabehistorien für s entsprechend deren Verhaltens klassifizieren. Diese Subfunktion sei hier o. B. d. A. mit g bezeichnet. Anhand der nachfolgend definierten Teilmengen von $\mathsf{dom}(s)$ wird das Zusammenwirken von g mit anderen in s eingebetteten Funktionen als Beziehung formuliert.

Definierte Eingaben eines Supersystems Wir betrachten zunächst die Menge aller Eingaben, welche, eingeschränkt auf die Schnittstelle I_g, keine Teilmenge der Domäne von g ist.

Definition 4.2 (P_0 – Die Menge der undefinierten Eingaben)

Die Menge aller Eingaben in s, die eingeschränkt auf I_g in g nicht definiert sind, wird mit $P_0^s(g)$ bezeichnet.

$$P_0^s(g) = \{x \in \mathit{dom}(s) : (x \,|\, I_g) \notin \mathit{dom}(g)\}$$

□

$P_0^s(g)$ beinhaltet unter anderem Eingaben für Funktionen, welche von g verschieden und ebenfalls in s eingebettet sind. Die Eingaben sind dabei entweder grundsätzlich auf anderen Kanälen als I_g zu beobachten. Oder es gibt zwar Eingaben auf Kanälen von I_g, diese sind jedoch in g nicht definiert. In dem Fall besteht die Möglichkeit einer Beziehung zu einer anderen eingebetteten Funktion f derart, dass ein weiterer Zugang zu g angeboten wird. Wir kommen darauf noch einmal bei der Definition des Beziehungstyps extend zurück.

Als Komplement zu $P_0^s(g)$ legen wir nun die Menge aller Eingaben in s fest, welche, eingeschränkt auf die Schnittstelle I_g, eine Teilmenge der Domäne von g ist. Sie beinhaltet folglich alle gültigen (engl. „valid“) Eingaben in g. Daher verwenden wir als Index für diese Menge den Buchstaben „V“.

Definition 4.3 (P_V- Die Menge der gültigen Eingaben)

Die Menge aller Eingaben in s, die eingeschränkt auf I_g in g definiert sind, wird mit $P_V^s(g)$ bezeichnet.

$$\begin{aligned} P_V^s(g) &= \{x \in \mathit{dom}(s) : (x \,|\, I_g) \in \mathit{dom}(g)\} \\ &= \mathit{dom}(s) \backslash P_0^s(g) \end{aligned}$$

Die Menge $P_V^s(g)$ enthält genau die Eingaben aus der Domäne von s, die nicht in $P_0^s(g)$ liegen. □

Undefinierte Eingaben eines Supersystems Man beachte, dass wir uns hier bei der Partitionierung der Eingaben ausschließlich auf die Domäne eines Supersystems s konzentrieren.

$$\mathsf{dom}(s) = P_0^s(g) \cup P_V^s(g)$$

Für alle anderen Eingaben aus $\{x \in \overrightarrow{I_s} : x \notin \mathsf{dom}(s)\}$ lassen wir insbesondere nicht zu, dass die auf die Schnittstelle I_g eingeschränkten Eingaben in der Domäne von g liegen. Denn für alle $x \in \overrightarrow{I_s}$ gilt:

$$x \notin \mathsf{dom}(s) \Rightarrow (x \,|\, I_g) \notin \mathsf{dom}(g)$$

Aus der Tatsache, dass es Eingaben gibt, die weder in der Domäne von s noch in der Domäne von g liegen, kann kein besonderer Zusammenhang zu anderen in s eingebetteten Funktionen abgeleitet werden.

Der Vollständigkeit halber sei aber erwähnt, dass wenn es ein $x \notin \mathsf{dom}(s)$ und $(x \,|\, I_g) \in \mathsf{dom}(g)$ geben sollte, es eine Funktion in s geben muss, die mit g nicht kombinierbar ist. Denn wenn $x \notin \mathsf{dom}(s)$ ist, dann ist insbesondere $(x \mid I_g) \notin \mathsf{dom}(s \dagger (I_g \blacktriangleright O_g))$. Wenn nun aber $(x \,|\, I_g) \in \mathsf{dom}(g)$, so ist $\mathsf{dom}(g) \not\subseteq \mathsf{dom}(s \dagger (I_g \blacktriangleright O_g))$ und g wäre somit keine Subfunktion von s.

Generelle Verhaltensvarianten

Wir konzentrieren uns nun wieder auf die in s definierten Eingaben und vergegenwärtigen uns die verschiedenen Perspektiven bezüglich einer betrachteten Funktion g. Ohne zunächst eine Abhängigkeit zu einer speziellen Funktion zu berücksichtigen, untersuchen wir folgende vier Verhaltensvarianten, die sich aus den entsprechenden Gegenüberstellungen ergeben.

- **Vorgabegemäßes Verhalten** (bei Restriktion auf die Schnittstelle von g)

  Für alle Eingabehistorien $x \in \mathsf{dom}(g)$ gilt $g.x = s\dagger(I_g \blacktriangleright O_g).x$. Wir schreiben in diesem Fall auch $g = s\dagger(I_g \blacktriangleright O_g)$.

- **Abweichendes Verhalten** (bei Restriktion auf die Schnittstelle von g)

  Es gibt eine Eingabehistorie $x \in \mathsf{dom}(g)$ mit $g.x \neq s\dagger(I_g \blacktriangleright O_g).x$. Hier schreiben wir auch kurz $g \neq s\dagger(I_g \blacktriangleright O_g)$.

- **Ausgabebeständiges Verhalten** (trotz uneingeschränkter Eingabekanalmenge)

  Für alle Eingabehistorien $x \in P^s_V(g)$ gilt $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = (s.x) \,|\, O_g$.

- **Ausgabeveränderliches Verhalten** (bei uneingeschränkter Eingabekanalmenge)

  Es gibt eine Eingabehistorie $x \in P^s_V(g)$ mit $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \neq (s.x) \,|\, O_g$.

Während sich die ersten beiden Eigenschaften auf die Konformität des Verhaltens zu der Vorgabe in g beziehen, zielen die beiden letzten auf das Schnittstellenverhalten unter Berücksichtigung von zusätzlichen Nachrichten auf anderen Kanälen ab. Bei den bereits eingeführten Mengen $P^s_0(g)$ und $P^s_V(g)$ war allein die Gültigkeit der Eingabe das zugrunde liegende Kriterium. Um mögliche Seiteneffekte beschreiben zu können, die gegebenenfalls auch durch Eingaben anderer Funktionen hervorgerufen werden, beziehen wir das Schnittstellenverhalten mit ein und partitionieren $P^s_V(g)$ entsprechend.

Definition 4.4 (P_S – die Menge der vorgabegemäßen Eingaben)

Die Menge aller Eingaben in s, bei denen die Ausgaben der Vorgabe in g entsprechen, wird mit $P^s_S(g)$ bezeichnet.

$$P^s_S(g) = \{x \in P^s_V(g) : g.(x \,|\, I_g) = s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)\}$$

□

Das Verhalten für die auf I_g eingeschränkten Eingaben entspricht dem in g vorgegebenem Verhalten. Die dort spezifizierten Ausgaben bleiben sozusagen bestehen (engl. „steady“). Wir indizieren daher diese Menge mit „S“. Im Komplement zu $P^s_S(g)$ finden sich folglich alle Eingaben in s, welche in abweichende Ausgaben resultieren. Diese Menge erhält den Index „D“ (engl. „deviate“).

Definition 4.5 (P_D – die Menge der abweichenden Eingaben)

Die Menge aller Eingaben in s, bei denen die Ausgaben nicht der Vorgabe in g entsprechen, wird mit $P_D^s(g)$ bezeichnet.

$$\begin{aligned} P_D^s(g) &= \{x \in P_V^s(g) : g.(x \,|\, I_g) \neq s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)\} \\ &= P_V^s(g) \backslash P_S^s(g) \end{aligned}$$

□

Selbstverständlich können in $P_V^s(g)$ insbesondere auch solche Eingaben enthalten sein, die Nachrichten auf anderen Kanälen als I_g aufweisen. Um mögliche Seiteneffekte durch solche Nachrichten identifizieren und beschreiben zu können, klassifizieren wir noch die Eingaben in $P_V^s(g)$ wie folgt.

Definition 4.6 (P_N– Die Menge der neutralen Eingaben)

Die Menge aller Eingaben in s, bei denen die Ausgaben von g ausschließlich von Nachrichten auf den mit I_g spezifizierten Kanälen abhängen beziehungsweise nicht von Nachrichten auf anderen Kanälen als I_g beeinflusst werden, wird mit $P_N^s(g)$ bezeichnet.

$$P_N^s(g) = \{x \in P_V^s(g) : s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = (s.x) \,|\, O_g\}$$

□

Die Menge $P_N^s(g)$ enthält die nicht beeinflussenden oder die neutralen Eingaben, das durch den Index „N“ zum Ausdruck gebracht wird. Die Einschränkung auf Eingaben aus der Domäne von g beziehungsweise auch das Ausblenden von Nachrichten auf anderen Kanälen hat keine Auswirkung auf das eingebettete Verhalten von g. Nachrichten auf anderen Kanälen als I_g beeinflussen das Verhalten nicht.

Im Gegensatz dazu bilden wir die Menge der beeinflussenden (engl. „affect“) Eingaben. Wir verwenden daher den Index „A“, um diese Menge zu bezeichnen.

Definition 4.7 (P_A– Die Menge der beeinflussenden Eingaben)

Die Menge aller Eingaben in s, bei denen die Ausgaben von g nicht nur von Nachrichten auf den mit I_g spezifizierten Kanälen abhängen, wird mit $P_A^s(g)$ bezeichnet.

$$\begin{aligned} P_A^s(g) &= \{x \in P_V^s(g) : s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \neq (s.x) \,|\, O_g\} \\ &= P_V^s(g) \backslash P_N^s(g) \end{aligned}$$

□

Obwohl die Eingaben in der Domäne von g liegen, wird das Verhalten durch Seiteneffekte, verursacht durch Eingaben auf anderen Kanälen, beeinflusst. Um diese Kanalmenge von den spezifizierten Eingabekanälen unterscheiden zu können, bezeichnen wir sie in Anlehnung an die Menge $P_A^s(g)$ mit $I_A(g)$.

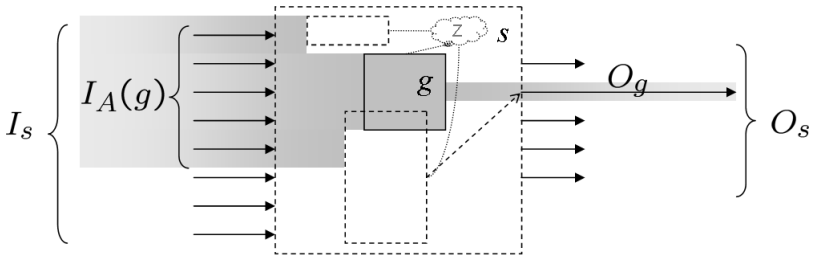


Abbildung 4.7: Die Kanäle aller beeinflussenden Nachrichten von g

Definition 4.8 (Die Kanalmenge I_A aller beeinflussenden Nachrichten)

Die Menge aller Kanäle, über die Nachrichten übertragen werden, welche g beeinflussen, wird mit $I_A(g)$ bezeichnet. Es gilt:

$$I_g \subseteq I_A(g) \subseteq I_s$$

□

In Abbildung 4.7 ist der Zusammenhang zwischen $I_A(g)$ und I_g beziehungsweise I_s veranschaulicht. Da die Eingabeschnittstelle mit der Definition der Funktion g festgelegt wird, beeinflussen die Nachrichten, welche in I_g liegen, selbstverständlich auch das Verhalten von g. Es gilt daher insbesondere $I_g \subseteq I_A(g)$.

I_s ist die Eingabeschnittstelle eines Supersystems s, in das g eingebettet ist. Sie entspricht der Vereinigungsmenge der Eingabeschnittstellen aller in s eingebetteten Funktionen und umfasst somit insbesondere auch die Eingabeschnittstelle von g. $I_A(g)$ ist also darüber hinaus stets eine Teilmenge von I_s, also $I_A(g) \subseteq I_s$. Im Übrigen ist $P_A^s(g)$ genau dann nicht leer, wenn $I_g \subset I_A(g)$ ist.

4.4.3 Interpretationen von Verhaltensmustern einer eingebetteten Funktion

Die Klassifizierung der Eingaben bezüglich einer betrachteten Funktion spiegelt die oben angeführten Verhaltensvarianten wider. Die Konformität der Ausgaben sowie die – je nach Berücksichtigung der Eingabekanäle – mögliche Abweichung der Ausgaben, sind

die dabei zugrunde liegenden Kriterien. Da diese Mengen klassifizierter Eingaben nicht notwendigerweise disjunkt sind, ergeben sich nun aus der Kombination der Verhaltensvarianten folgende Fälle, die wir anknüpfend im Einzelnen diskutieren.

- Vorgabegemäßes und unbeeinflussbares Verhalten
- Vorgabegemäßes, aber beeinflussbares Verhalten
- Unabdingbar abweichendes Verhalten
- Nicht vorgabegemäßes, aber beeinflussbares Verhalten

Vorgabegemäßes und unbeeinflussbares Verhalten

Das Verhalten von g in s entspricht stets den Vorgaben. Insbesondere wird das Verhalten von g auch nicht von anderen in s eingebetteten Funktionen über Eingaben auf anderen Kanälen als I_g beeinflusst.

$$\begin{aligned} & g = s\dagger(I_g \blacktriangleright O_g) \\ \textit{und} \quad & \forall\, x \in \mathsf{dom}(s) : g.(x \,|\, I_g) = (s.x) \,|\, O_g \end{aligned}$$

Sollte es auch gemeinsame Eingaben mit einer anderen Funktion in s geben, hat auch dies keinen Effekt für das Verhalten von g. Wir sprechen deshalb hier von einer „*unbeeinflussten Einbettung*“ von g in s, wenn gilt:

$$\begin{aligned} & P^s_V(g) = P^s_S(g) = P^s_N(g) \\ \text{beziehungsweise} \quad & P^s_D(g) = \varnothing \wedge P^s_A(g) = \varnothing \end{aligned}$$

Speziell entspricht hier die Kanalmenge der beeinflussenden Eingaben den Eingabekanälen von g, das heißt $I_A(g) = I_g$.

Vorgabegemäßes, aber beeinflussbares Verhalten

Das Verhalten von g in s entspricht stets der Vorgabe, solange nur Eingaben über die spezifizierten Eingabekanäle erfolgen. Es gibt allerdings Situationen, bei denen die Ausgaben auf O_g abweichen, obwohl die Eingaben über I_g der Vorgabe in g entsprechen.

$$\begin{aligned} & g = s\dagger(I_g \blacktriangleright O_g) \\ \textit{und} \quad & \exists\, x \in P^s_V(g) : g.(x \,|\, I_g) \neq (s.x) \,|\, O_g \end{aligned}$$

Die Beeinflussung durch eine andere eingebettete Funktion in s über gemeinsame Eingaben kann in diesem Fall ausgeschlossen werden, da wir $g = s\dagger(I_g \blacktriangleright O_g)$ angenommen haben. Ob nun g gemeinsame Eingabekanäle mit anderen Funktionen hat oder auch nicht, spielt in dem Fall keine Rolle. Bleibt folglich nur die Beeinflussung aufgrund von Nachrichten auf anderen Eingabekanälen als I_g.

Sobald Eingaben auf weiteren Kanälen anderer Funktionen zugelassen werden, können die Ausgaben von den Vorgaben in g abweichen. Demgemäß ist die Menge der Kanäle,

über die Nachrichten übertragen werden, die das Verhalten von g in s beeinflussen, größer als I_g.

$$I_g \subset I_A(g)$$

Generell sind nun zwei Mengen verschieden, wenn ein Element der einen Menge kein Element der anderen Menge ist, oder umgekehrt. Für eine Eingabe $x \in P^s_V(g)$ kommen für die Ungleichheit der Ausgabemengen daher drei sinnvolle Ausprägungen in Frage.

- $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$ ist eine echte Teilmenge von $(s.x) \,|\, O_g$.
- $(s.x) \,|\, O_g$ ist eine echte Teilmenge von $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$, mit anderen Worten: die Ausgabemenge reduziert sich.
- Es gibt eine Historie in $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$, die nicht in $(s.x) \,|\, O_g$ enthalten ist, und umgekehrt. Das heißt, die Ausgaben sind nicht konform zur Vorgabe.

Die erste erwähnte Möglichkeit, dass $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$ eine echte Teilmenge von $(s.x) \,|\, O_g$ ist, dass es also zusätzliche Ausgaben gibt, wenn mehr Eingabekanäle betrachtet werden, ist nicht realistisch. Im Gegenteil - vielmehr kann die Restriktion der Eingabekanäle zu nichtdeterministischem Verhalten führen. Einen derartigen Zusammenhang schließen wir deshalb aus.

Reduzierung der Ausgaben

Dagegen ist es vorstellbar, dass gerade das Einbeziehen von Eingaben auf zusätzlichen Kanälen die Menge der möglichen Ausgaben reduziert. Wir nehmen hier an, dass das Verhalten von g beziehungsweise von $s\dagger(I_g \blacktriangleright O_g)$ bereits nichtdeterministisch ist und es eine weitere Funktion in s gibt, die zur Präzisierung des Verhaltens beiträgt. Da die Beeinflussung über gemeinsame Eingaben unter den gegebenen Bedingungen auszuschließen ist, kann diese nur über Eingaben auf anderen Kanälen als I_g erreicht werden. Ist diese Funktion mit f bezeichnet, so gilt

$$I_A(g) \cap I_f \neq \varnothing$$

Dass durch die Kombination mit einer anderen Funktion das Verhalten durch Reduktion der Ausgabevielfalt präzisiert wird, ist nicht ungewöhnlich. Diesen Zusammenhang werden wir noch als refine Beziehung zwischen Funktionen kennen lernen.

Rein formal ist im Extremfall zwar auch die leere Ausgabemenge für $(s.x) \mid O_g$ möglich. Dies bedeutete aber die Undefiniertheit der Eingabe x, was wir hier aber ausschließen wollen. Üblicherweise werden vielmehr Nachrichten ausgegeben, die jedoch bei der Projektion auf O_g ausgeblendet werden. Diese Situation wird nachfolgend skizziert.

Nicht konforme Ausgaben

Es sei daran erinnert, dass es hier um Abweichungen von den spezifizierten Ausgaben geht, die ihre Ursache in Eingaben auf weiteren Kanälen anderer Funktionen haben. Diese Situation ist typischerweise auch bei der Deaktivierung einer Funktion durch eine andere Funktion gegeben (disable).

Im Falle der Deaktivierung einer betrachteten Funktion g gibt es eine Funktion in s, die durch bestimmte Nachrichten den Zustand von s derart manipuliert, dass insbesondere auch definitionsgemäße Eingaben von g in Ausgaben resultieren, die verschieden von den in g spezifizierten Ausgaben sind. Die Ausgabehistorien stimmen hier nicht oder nur teilweise[1] mit der Vorgabe überein.

Zwar werden durchaus Nachrichten als Reaktion auf eine Eingabe x ausgegeben, möglicherweise allerdings auf anderen Kanälen als O_g. Diese werden dann gerade bei der Projektion auf O_g ausgeblendet, und keine Ausgabe ist auf diesen Kanälen zu beobachten. Genauer gesagt, das Resultat der Projektion auf O_g ist dann eine Menge leerer Nachrichtenströme.

Kommen wir nun zu den Konstellationen, bei denen das Schnittstellenverhalten von der Vorgabe abweicht.

Unabdingbar abweichendes Verhalten

Das Verhalten weicht von der Vorgabe in g ab, und zwar unabhängig davon, ob Eingaben auf zusätzlichen Kanälen in Betracht gezogen werden.

$$\begin{aligned} & g \neq s\dagger(I_g \blacktriangleright O_g) \\ \textit{und} \quad & \forall\, x \in P^s_V(g) : s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = (s.x) \,|\, O_g \end{aligned}$$

Beziehungsweise von der anderen Seite aus betrachtet: gleichgültig ob Eingaben auch über andere Kanäle als I_g erfolgen, die Ausgaben von g in s sind davon unberührt, wobei es allerdings eine Eingabe $x \in P^s_V(g)$ gibt, die in nicht vorgabegemäße Ausgaben resultiert.

Auch hier werden wir die Ungleichheit der Ausgabemengen differenziert betrachten. Zunächst entspricht aber bei dieser Eingabe x das Schnittstellenverhalten nicht den Vorgaben in g, selbst wenn die Eingaben von anderen in s eingebetteten Funktionen berücksichtigt werden. Hier gilt daher allgemein

$$I_A(g) = I_g$$

[1] Bei Ausgabemengen mit jeweils nur einer Historie kann auf die „teilweise Übereinstimmung“ verzichtet werden. Die Historien sind dort entweder identisch oder verschieden. Beinhalten die Mengen jeweils mehrere Historien, kann mathematisch betrachtet die Ungleichheit der Mengen auch bedeuten, dass nur eine Teilmenge von Historien übereinstimmt.

Der Vollständigkeit halber sei ebenfalls erwähnt, dass hier insbesondere

$$g.(x \,|\, I_g) \neq (s.x) \,|\, O_g$$

gilt. Aber welche Ursachen sind nun für dieses unabdingbar abweichende Schnittstellenverhalten für diese Eingabe x denkbar?

Wie bereits oben erwähnt, kann dies einerseits an der Regelung des Zugangs zu g liegen, wenn im Rahmen der Integration von Funktionen das Gesamtverhalten von s mittels geeigneter Systemzustände festgelegt wird. Diese Situation können wir aber ausschließen, da sonst der Zugang zu g durch die Nutzung anderer Funktionen in s wieder ermöglicht werden sollte, was hier augenscheinlich nicht gegeben ist.

Auf der anderen Seite kann sich der Einfluss einer anderen eingebetteten Funktion in s auf das Schnittstellenverhalten auswirken, und zwar dann, wenn die Eingabekanäle der Funktionen nicht disjunkt sind. Bei der gegebenen Sachlage muss es tatsächlich solch eine Funktion in s geben, die wir im Folgenden o. B. d. A. mit f bezeichnen. Diese Funktion f besitzt also gemeinsame Eingaben mit g ($I_f \cap I_g \neq \varnothing$) und ist für das im Vergleich zu der Vorgabe in g abweichende Schnittstellenverhalten verantwortlich.

Im Übrigen muss es bei gemeinsam definierten Eingaben nicht notwendigerweise zu Ausgabekonflikten kommen. Zum Beispiel können die Ausgabekanäle von f und g ohne weiteres disjunkt sein, sprich: $O_f \cap O_g = \varnothing$. In diesem Fall sind die Ausgaben von f zwar für die Ausgaben auf O_g nicht relevant. Sie tragen jedoch dazu bei, den Zustand von s so zu beeinflussen, dass abweichende Ausgaben zustande kommen. Wie im vorher gehenden Fall untersuchen wir die Abweichungen im Detail. Die prinzipiellen Ausprägungen sind auch hier:

- An der Schnittstelle entstehen zusätzliche Ausgaben.
- Die Menge der Ausgaben wird reduziert.
- Die Ausgaben stimmen nicht oder nur teilweise überein.

Zusätzliche Ausgaben

Wenn zusätzliche Ausgaben erzeugt werden,

$$g.(x \,|\, I_g) \subset s \dagger (I_g \blacktriangleright O_g).(x \,|\, I_g)$$

muss eine andere Funktion ebenfalls Nachrichten auf O_g ausgeben. Die Disjunktheit der Ausgabekanäle der Funktionen ist somit ausgeschlossen.

Die Vergrößerung der Ausgabemenge ist das Ergebnis unterschiedlicher Ausgabemengen. Die Kombination von g mit der anderen Funktion verhält sich für diese Eingabe x dann nicht deterministisch. Um dies zu verhindern zu können, werden wir eine exclude Beziehung benötigen, welche explizit die simultane Nutzung mit einer anderen Funktion ausschließt.

Reduzierung der Ausgaben

Wir nehmen hier an, dass sich die betrachtete Funktion g bereits nicht deterministisch verhält. Eine andere in s eingebettete Funktion trage nun dazu bei, die Menge der Ausgaben zu reduzieren.

$$s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \subset g.(x \,|\, I_g)$$

Ob die Funktionen hierbei gemeinsame Ausgabekanäle besitzen, spielt keine Rolle. Selbst wenn die verantwortliche Funktion Nachrichten auf O_g ausgeben sollte, stünden diese dann nicht im Widerspruch zu den vorgegeben. Auch in diesem Fall werden wir von einer refine Beziehung zwischen den Funktionen sprechen (vergleiche Absatz unter „Vorgabegemäßes, aber beeinflussbares Verhalten“ auf Seite 82). Der Unterschied zu vorher liegt darin, dass die Ausgabevielfalt nun aufgrund von gemeinsam definierten Eingaben reduziert wird.

Auch hier schließen wir die leere Ausgabemenge für $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$ aus, da die Undefiniertheit der Eingabe x im Normalfall nicht gegeben ist.

Andere Ausgaben

In der Kombination mit einer anderen in s eingebetteten Funktion werden die spezifizierten Eingaben von g zwar akzeptiert,

$$s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \neq \varnothing$$

die Ausgabehistorien stimmen jedoch nicht oder nur teilweise mit der Vorgabe überein (siehe dazu auch die Fußnote auf Seite 83). Auch hier untersuchen wir den Effekt bei gemeinsamen oder unterschiedlichen Ausgabekanälen der Funktionen.

Sei diese andere Funktion wieder o. B. d. A. mit f bezeichnet. Falls nun f und g gemeinsame Ausgabekanäle besitzen, generiert f zwar Ausgaben auf O_g, diese sind jedoch nicht vorgabegemäß. Man beachte, dass es sich hier nicht um zusätzliche Ausgabehistorien, sondern gleichsam um das Ersetzen von Ausgabehistorien handelt. Wie eingangs erklärt, geht es hier um Abweichungen von den spezifizierten Ausgaben, die ihre Ursache in gemeinsamen Eingaben haben. Wenn die Ersetzung von Ausgaben nicht erwünscht ist, sollte die simultane Nutzung der Funktionen ausgeschlossen werden(exclude). Sonst führt dieser Zusammenhang zu einem neuen Beziehungstyp, der später mit replace betitelt wird. Der Funktion f wird in diesem Fall eine höhere Priorität als g beigemessen.

Bei disjunkten Ausgabekanälen der Funktionen kommt nur die Möglichkeit in Betracht, dass zwar durchaus Nachrichten als Reaktion auf x ausgegeben werden, allerdings auf anderen Kanälen als O_g. Diese werden, wie bereits erwähnt, bei der Projektion auf O_g ausgeblendet, und auf den Kanälen O_g ist keine Ausgabe zu beobachten.

Rein formal ist auch die leere Ausgabe (Ausgabehistorie mit leeren Nachrichtenströmen) eine zusätzliche Ausgabe, die im Widerspruch zur spezifizierten Ausgabe steht. In der Kombination von f und g bewirkt nun die Priorisierung von f gegenüber g die Auflösung dieses Ausgabekonflikts. Die Entstehung nichtdeterministischen Verhaltens kann verhindert werden, indem die ursprünglich spezifizierte Ausgabe von g in s keine Option mehr ist.

Nicht vorgabegemäßes, aber beeinflussbares Verhalten

In der abschließend diskutierten Verhaltensvariante kommen abweichende Ausgaben sowohl bei uneingeschränkter Eingabekanalmenge, als auch bei der Restriktion auf die Schnittstelle von g vor.

$$\begin{aligned} & g \neq s\dagger(I_g \blacktriangleright O_g) \\ \textit{und} \quad & \exists\, x \in P^s_V(g) : s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \neq (s.x) \,|\, O_g \end{aligned}$$

Man beachte, dass die Ungleichheit der Ausgaben jeweils durch eine andere Eingabe verursacht werden kann. Das Verhalten muss daher im Detail untersucht und die Gleichheit beziehungsweise die Verschiedenheit der Ausgaben für einzelne Eingaben überprüft werden. Definitionsgemäß existiert zunächst eine Eingabe $x_D \in P^s_D(g)$ und eine Eingabe $x_A \in P^s_A(g)$, wobei nicht notwendigerweise $x_D \in P^s_A(g)$ oder $x_A \in P^s_D(g)$ gilt.

Im Falle $x_A \notin P^s_D(g)$ muss folglich $x_A \in P^s_S(g)$ und damit $g.(x_A \,|\, I_g) = s\dagger(I_g \blacktriangleright O_g).(x_A \,|\, I_g)$ sein, was bereits unter dem Absatz „Vorgabegemäßes, aber beeinflussbares Verhalten“ diskutiert wurde (siehe Seite 81).
Wenn umgekehrt $x_D \notin P^s_A(g)$ gilt, muss logischerweise $x_D \in P^s_N(g)$ sein. Auch diese Situation mit $s\,\dagger\,(I_g \blacktriangleright O_g).(x_D \mid I_g) = (s.x_D) \mid O_g$ wurde bereits unter „Unabdingbar abweichendes Verhalten“ auf Seite 83 erörtert.

Schließlich bleibt noch zu untersuchen, dass es tatsächlich eine Eingabe x gibt, die sowohl ein Element der Menge der abweichenden Eingaben, als auch ein Element der Menge der beeinflussenden Eingaben ist. Es existiert also ein

$$x \in (P^s_A(g) \cap P^s_D(g))$$

mit $g.(x \,|\, I_g) \neq s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$ und $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) \neq (s.x) \,|\, O_g$.

Wir unterscheiden hierbei die beiden Fälle, ob wenigstens die Berücksichtigung von Eingaben auf zusätzlichen Kanälen zu einem Verhalten führt, das der Vorgabe entspricht, oder selbst dies keine Auswirkung hat.

Vorgabegemäßes Verhalten durch Beeinflussung

Im Gegensatz zur vorigen Konstellation mit unabdingbar abweichendem Verhalten kann hier eine Zugangsregelung zu g durchaus eine entscheidende Rolle spielen.

$$x \in (P_A^s(g) \cap P_D^s(g)) \quad \wedge \quad g.(x \,|\, I_g) = (s.x) \,|\, O_g$$

Wenn g etwa standardmäßig deaktiviert ist, weicht per Voreinstellung die Ausgabe ab und erst wenn ein bestimmter Systemzustand erreicht ist, erfolgen die Ausgaben wie spezifiziert. Um in diesen Zustand zu gelangen, bedarf es dann im Allgemeinen der Nutzung einer anderen eingebetteten Funktion, welche die deaktivierte Funktion frei schaltet. Zumindest aber sind zusätzliche Eingaben auf anderen Eingabekanälen als I_g erforderlich.

Sobald Eingaben auf weiteren Kanälen anderer Funktionen zugelassen werden, entsprechen die Ausgaben auch den Vorgaben in g. Nur korrekte Eingaben an der Schnittstelle von g beziehungsweise nur wie in g spezifizierte Eingaben reichen nicht aus. Diese Situation ist, wie erwähnt, typischerweise bei der Aktivierung einer Funktion durch eine andere Funktion gegeben (enable).

Eine andere Variante, bei der zusätzlich berücksichtigte Eingaben auf anderen Kanälen als I_g zu vorgabegemäßem Verhalten führen, kann auch bei der Priorisierung von Funktionen verzeichnet werden. Solch eine Situation haben wir bereits im Zusammenhang mit widersprüchlichen Ausgaben kombinierter Funktionen kennen gelernt (siehe Seite 85). Eine etwaige Voreinstellung des Systemzustands ist hier allerdings nicht mehr relevant, sondern ausschließlich die eingegebenen Nachrichten. Angenommen es gibt eine Eingabehistorie

$$x \in (P_A^s(g) \cap P_D^s(g)) \quad \wedge \quad g.(x \,|\, I_g) \subset s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$$

sprich: es muss eine weitere Funktion in s geben, welche wir o. B. d. A. mit f bezeichnen, deren Ausgaben im Widerspruch zu denen von g stehen. Gilt nun darüber hinaus aber

$$g.(x \,|\, I_g) = (s.x) \,|\, O_g$$

muss es folglich Eingaben auf zusätzlichen Kanälen geben, die dazu führen, dass sich g in s vorgabegemäß verhält. Eine denkbare Interpretation ist hier die Vermeidung widersprüchlicher Ausgaben durch Eingaben auf zusätzlichen Kanälen, indem der betrachteten Funktion g eine höhere Priorität beigemessen wird als f. Die Ausgaben von g ersetzen in diesem Fall die Ausgaben von f (replace).

Man beachte, dass wenn gemeinsame Eingaben von f bereits Einfluss auf das Verhalten von g haben, dann der Einfluss im Allgemeinen auch nicht durch die Berücksichtigung von Eingaben auf zusätzlichen Kanälen gleichsam neutralisiert wird. Die

oben vorher beschriebene Festlegung der Priorität einer Funktion ist sicher eine Lösung. Ein anderer Weg besteht darin, dass es zum Beispiel eine Funktion h in s gibt, welche f deaktiviert. Dies führte an dieser Stelle aber zu einer Diskussion komplexerer Abhängigkeiten, die vor dem Hintergrund eines Beziehungskalküls abgeleitet werden müssten. Wir konzentrieren uns hier aber auf elementare Zusammenhänge zwischen Funktionen.

Unbeeinflussbar abweichendes Verhalten

Das Schnittstellenverhalten für die Eingabe x entspricht nicht der Vorgabe, gleichgültig ob Nachrichten nur auf den in g spezifizierten Eingabekanälen oder auch auf allen zusätzlichen Kanälen der anderen eingebetteten Funktionen betrachtet werden.

$$x \in (P_A^s(g) \cap P_D^s(g)) \quad \wedge \quad g.(x \,|\, I_g) \neq (s.x) \,|\, O_g$$

Hier haben Nachrichten auf zusätzlichen Eingabekanälen zwar Einfluss auf das Verhalten. Doch auch diese führen zu keinen vorgabegemäßen Ausgaben.

Rufen wir uns wieder die Situation in Erinnerung, bei der durch die Kombination von Funktionen widersprüchliche Ausgaben entstehen und für eine Eingabehistorie $x \in (P_A^s(g) \cap P_D^s(g))$ wiederum gelte

$$g.(x \,|\, I_g) \subset s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$$

Wenn nun aber im Gegensatz zu oben, auch unter Berücksichtigung aller Eingabekanäle, das Verhalten nicht vorgabegemäß ist

$$g.(x \,|\, I_g) \neq (s.x) \,|\, O_g \quad \wedge \quad (s.x) \,|\, O_g \subset s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$$

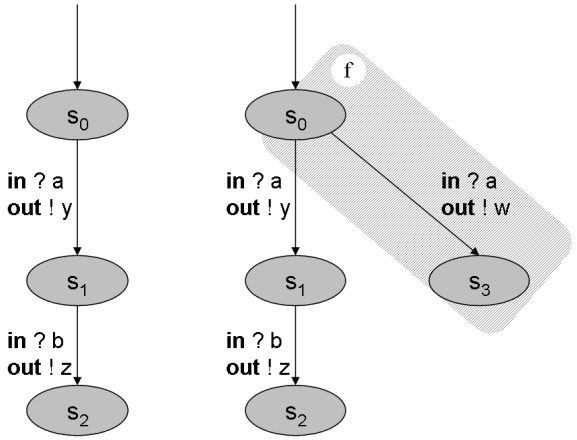
kann man auch bei dieser Sachlage von einer Priorisierung ausgehen. Wir interpretieren hier die Vermeidung widersprüchlicher Ausgaben durch Eingaben auf zusätzlichen Kanälen. Diese Eingaben lassen die Ausgaben von g nicht zu und messen damit der betrachteten Funktion g eine niedere Priorität bei. In diesem Fall ersetzen die Ausgaben einer anderen Funktion die Ausgaben von g (replace).

Da wir hier Mengen von Ausgabehistorien betrachten, bieten sich theoretisch noch mehr Möglichkeiten an, deren Ungleichheit zu interpretieren und entsprechend zu kombinieren. Allerdings führte dies hier zu keiner neuen Erkenntnis hinsichtlich der Zusammenhänge zwischen Nutzungsfunktionen.

4.5 Kombinierbarkeit von Funktionen

In den vorigen Abschnitten stößt man an der einen oder anderen Stelle bereits auf Formulierungen wie „Funktionen sind nicht kombinierbar“, „konsistente Kombination“ oder auch die „Kombinierbarkeit von Funktionen“. Während die *Kombination* von Funktionen bereits in Abschnitt 3.3 eingeführt wurde, wurde der Begriff der *Kombinierbarkeit* bisher weitgehend informell verwendet. Beide Begriffe hängen selbstverständlich zusammen, etwa in dem Sinne, dass kombinierbare Funktionen tatsächlich kombiniert werden können. Aber was bedeutet es für das Verhalten eines Softwaresystems, wenn nicht kombinierbare Funktionen kombiniert werden?

Offenbar sollten beide Begriffe nicht synonym verwendet werden. In diesem Abschnitt untersuchen wir deren Zusammenhang im Detail und definieren die Kombinierbarkeit von Funktionen formal. Wir beziehen dafür neben der betrachteten Funktion g eine andere zu kombinierende Funktion f in die Überlegungen mit ein. Sehen wir uns zunächst ein Beispiel für die Kombination zweier fiktiver Funktionen an.



(a) Funktion g (b) In Kombination mit Funktion f

Abbildung 4.8: Inkonsistente Spezifikation kombinierter Funktionen

Wie aus den Zustandsübergangsdiagrammen in Abbildung 4.8 ersichtlich, besitzen g und f eine gemeinsame Eingabe ($\langle a \rangle$), deren Ausgaben allerdings nicht konsistent sind. In der folgenden Tabelle sind die widersprüchlichen Ausgaben zwischen den Symbolen ► ◄ gekennzeichnet.

$x \in \mathsf{dom}(s)$	$s.x$		$s \dagger (I_g \blacktriangleright O_g).(x \mid I_g)$	$g.(x \mid I_g)$	
$\langle a \rangle$	$\{\langle y \rangle, \langle w \rangle\}$	$\blacktriangleright$	$\{\langle y \rangle, \langle\,\rangle\}$	$\{\langle y \rangle\}$	$\blacktriangleleft$
$\langle a, b \rangle$	$\{\langle y, z \rangle\}$		$\{\langle y, z \rangle\}$	$\{\langle y, z \rangle\}$	

Kombinierbare Funktionen zeichnen sich also zunächst dadurch aus, dass sie keine widersprüchlichen Ausgaben besitzen. Denn gäbe es für bestimmte Eingaben jeweils unterschiedliche Nachrichten auf gleichen Ausgabekanälen, vergrößerte sich die Menge der Ausgabehistorien und beinhaltete sämtliche möglichen Ausgabevarianten. Das Verhalten einer Funktion ist dann nicht mehr deterministisch, wie in Abbildung 4.8(b) zu ersehen ist.

Werden die Funktionen im Rahmen einer Superfunktion kombiniert und projiziert man die Ausgaben dieser Superfunktion jeweils auf die Ausgabekanäle einer Funktion, so entstehen keine zusätzlichen Ausgabehistorien. Diese intuitiv einsichtige Eigenschaft des Zusammenhangs kombinierbarer Funktionen lässt sich mathematisch mithilfe des in Abschnitt 4.3 eingeführten subfunction Beziehungstyps (siehe Definition 4.1) folgendermaßen ausdrücken.

Definition 4.9 (Kombinierbarkeit von Funktionen)

Zwei Funktionen f und g sind kombinierbar genau dann, wenn es eine Superfunktion beziehungsweise ein Supersystem s gibt, welches den Zugang zu beiden Funktionen anbietet.

$$\begin{aligned} &f \text{ und } g \text{ kombinierbar} \\ \Leftrightarrow\ &\exists\, s : \boldsymbol{subfunction}(s, f) \wedge \boldsymbol{subfunction}(s, g) \end{aligned}$$

□

Zwei Funktionen sind genau dann kombinierbar, wenn der Zugang zu beiden Funktionen – ungeachtet möglicher Zusammenhänge zwischen den Funktionen – von einem Softwaresystem angeboten wird.

Triviale Kombinierbarkeit

Man beachte, dass die Äquivalenz in Definition 4.9 insbesondere für den Fall gilt, wenn die Funktionen f und g bereits in einer subfunction Beziehung stehen. Deren Kombinierbarkeit ist dann allerdings trivial, da dies bereits aus der Transitivität der Subfunktionsrelation folgt. Sei o. B. d. A. g eine Teilfunktion von f.

$$\mathbf{subfunction}(s, f) \wedge \mathbf{subfunction}(f, g) \Rightarrow \mathbf{subfunction}(s, f) \wedge \mathbf{subfunction}(s, g)$$

Für unsere Untersuchungen schließen wir diese triviale Kombinierbarkeit der Funktionen aus, und gehen davon aus, dass weder g eine Teilfunktion von f, noch f eine Teilfunktion

von g ist. Wenn daher im Folgenden von der *Kombinierbarkeit von Funktionen* gesprochen wird, wird implizit davon ausgegangen, dass diese *nicht trivial* ist.

Die Kombination kombinierbarer Funktionen

Wenn nun neben der betrachteten Funktion g eine weitere Funktion f in s ins Spiel gebracht wird und wir des Weiteren annehmen, dass f eine Teilfunktion von s ist,

$$\textbf{subfunction}(s, f)$$

interessieren hinsichtlich der Kombinierbarkeit der Funktionen folgende Fragestellungen:

- Gilt auch in der Kombination mit f immer noch **subfunction**(s, g)?
- Beziehungsweise unter welchen Bedingungen gilt diese Relation?

Im ungünstigsten Fall ist die Funktion g mit f nicht kombinierbar. Dann gibt es Ausgaben der Funktionen, die nicht miteinander vereinbar sind beziehungsweise im Widerspruch zueinander stehen. Kombinierte man die Funktionen dennoch ohne geeignete Maßnahmen im Sinne einer Einschränkung der Nutzung, käme es zwangsläufig zu „unwanted feature interactions“. Ein möglicher Zusammenhang zwischen den Funktionen, und zwar aus der Perspektive von g, zeigt sich am Verhalten von g in s.

1. Wenn g nicht im Geringsten von f beeinflusst wird, es also zu keinen Veränderungen der Ausgaben von g in s kommt, gleichgültig wie sich die jeweiligen Eingabehistorien überlagern, so ist g in ihrem Verhalten unabhängig von f. Um diesen Zusammenhang zu beschreiben, werden wir den Beziehungstyp independent verwenden, welcher eigens im nächsten Abschnitt 4.6 besprochen wird.

   Gilt im Übrigen auch umgekehrt, dass f in ihrem Verhalten unabhängig von g ist (die „wechselseitige Unabhängigkeit“ der Funktionen), so sind f und g stets kombinierbar.

2. Wenn die Verhaltensunabhängigkeit nicht gegeben ist, so gibt es mindestens jeweils eine Eingabehistorie von g und f, die sich nicht beliebig überlappen können, ohne eine entsprechende Verhaltensänderung im Vergleich zu der Vorgabe in g festzustellen.

   Die Kombinierbarkeit der Funktionen muss hier nicht notwendigerweise die gesamte Domäne eines Supersystems umfassen. Falls zum Beispiel Eingaben auf anderen Kanälen als I_g Einfluss auf das Verhalten haben, kann sich insbesondere die Kombinierbarkeit auch nur auf eine bestimmte Teilmenge der Domäne beschränken. Kann eine entsprechende Teilmenge gültiger Eingaben definiert werden, sprechen wir auch von einer „bedingten Kombinierbarkeit“ der Funktionen bezüglich dieser Teilmenge.

3. Es besteht ein Zusammenhang zwischen den Funktionen, der sich zwar nicht unmittelbar anhand veränderter Ausgaben bei der Projektion der Historien von s auf die Kanäle von g festmachen lässt, dennoch aber von den Nutzern der Funktionen als nicht unabhängig wahrgenommen wird.

Die Diskussion solcher Beziehungstypen, die sich auf die letzten beiden Aspekte beziehen, wird in Kapitel 5 fortgesetzt.

Die Kombination nicht kombinierbarer Funktionen

Zweifelsohne ist es ratsam, insbesondere beim Systemdesign nur kombinierbare Funktionen zu akzeptieren. Wie bereits unter 2. angedeutet, ist allerdings aus praktischen Überlegungen heraus ein etwas weiter gefasster Kombinierbarkeitsbegriff notwendig. So wollen wir etwa zulassen, Funktionen auch dann zu kombinieren, wenn sie auf einer Teilmenge der Domäne eines Supersystems kombinierbar sind. Auch unkritische oder gewollte Änderungen des Verhaltens sollten nicht der Grund dafür sein, Funktionen nicht zu kombinieren.

Würden in einem Softwaresystem Funktionen kombiniert werden, die zwar dieselben Eingaben aber verschiedene Ausgaben besitzen, führte dies ohne Ergreifen geeigneter Maßnahmen letztlich zu nichtdeterministischem Systemverhalten. Dies ist oft im Zusammenhang mit Produktvarianten und verschiedenen Ausprägungen einer Funktion zu finden. Typischerweise besitzen solche Funktionen zwar denselben Zugang, in der Standard-Variante sind die Ausgaben eher elementar – im Gegensatz zu Komfortausgaben einer Premium-Variante. Auch das Vorhandensein von zusätzlichen Funktionen, die etwa bei einem Standardsystem nicht vorgesehen sind, erlauben entsprechend aufwendigere Ausgaben. Bei einem Mobiltelefon können zum Beispiel Namen statt Nummern angezeigt werden, wenn eine Adressbuchfunktion vorhanden ist.

An diesen Beispielen wird der Unterschied von „kombinierten Funktionen“ und „kombinierbaren Funktionen“ besonders deutlich. Solche Funktionen wären im Prinzip nicht kombinierbar. Eine Möglichkeit ist der gegenseitige Ausschluss, um solche Funktionen dennoch innerhalb eines Softwaresystems zu kombinieren. Die simultane Nutzung solcher Funktionen wird so explizit ausgeschlossen und führt im Allgemeinen zur Nutzbarkeit der Funktionen in jeweils unterschiedlichen Betriebsarten eines Softwaresystems. Der dafür infrage kommende Beziehungstyp wird mit exclude bezeichnet.

Solche „überlagerten“ und per Definition nicht kombinierbare Funktionen können jedoch nicht nur per exclude kombiniert werden, sondern auch dann, wenn dafür Sorge getragen wird, dass die unterschiedlichen Ausgaben nicht beliebig, sondern kontrolliert übertragen werden. Dies kann einerseits dadurch erreicht werden, dass es gar nicht erst zu mehrdeutigen Ausgaben kommt, indem die Ausgabe einer Funktion eine höhere Priorität erhält (replace). Andererseits können nichtdeterministische Ausgaben umgangen werden, indem

gezielt – im Sinne einer Verfeinerung – eine (deterministische) Auswahl der Ausgabe etwa von einer dritten Funktion getroffen wird. Dazu verwenden wir den Beziehungstyp refine.

Beziehungen zwischen kombinierbaren und bedingt kombinierbaren Funktionen

Die entscheidende Randbedingung bei der Festlegung eines Beziehungstyps ist die Kombinierbarkeit der Funktionen. Sie steht als Zusammenhang zwischen Funktionen über allen anderen möglichen Beziehungstypen. Daher gibt es keine speziellen Beziehungstypen für nicht kombinierbare Funktionen – ihr Zusammenhang wird dann faktisch als „nicht kombinierbar“ bezeichnet. Solche Funktionen können niemals gemeinsam Teilfunktionen eines Softwaresystems sein.

Alle im Folgenden definierten Beziehungstypen charakterisieren daher Zusammenhänge von bedingt kombinierbaren oder unbedingt kombinierbaren Funktionen. Im Falle der bedingten Kombinierbarkeit gelten die Subfunktionseigenschaften nur für eine Teilmenge der Domäne eines Supersystems. Oder aber der Zugang zu einer der Funktionen ist nur dann gewahrt, wenn nicht nur die spezifizierten Nutzungskonventionen, also die definierten Interaktionsmuster, eingehalten werden, sondern beispielsweise zusätzliche Eingaben der anderen Funktion zwingend notwendig sind. Beziehungstypen beschreiben letztlich die Zusammenhänge kombinierter Funktionen beziehungsweise den Einfluss einer Funktion auf eine andere Funktion, wenn die Funktionen in ein Supersystem eingebettet sind.

4.6 Unabhängigkeit von Funktionen

Die Unabhängigkeit einer Funktion von einer anderen Funktion scheint nur auf den ersten Blick die offensichtlichste aller Beziehungen zwischen zwei Funktionen zu sein. Die vermeintlich auf der Hand liegende einfache Bedingung, dass sich die Spezifikationen zweier unabhängiger Funktionen nicht widersprechen dürfen, ist aber weder notwendig noch hinreichend.

Zunächst mag dies zwar ein nachvollziehbares Kriterium sein, welches sich in der Tat auf das kombinierte Verhalten der Funktionen bezieht, jedoch eher im Kontext der Kombinierbarkeit von Funktionen anzusiedeln ist. Darüber hinaus vernachlässigt man insbesondere, dass die Unabhängigkeit nicht notwendigerweise wechselseitig gilt und im – Gegensatz zur Kombinierbarkeit – keine symmetrische Relation ist. Denn selbst wenn eine betrachtete Funktion g unabhängig von einer Funktion f ist, kann f dagegen im Rahmen einer übergeordneten Superfunktion durchaus von g beeinflusst werden. Erst aus der wechselseitigen Unabhängigkeit der Funktionen kann auf deren Kombinierbarkeit geschlossen werden.

Wie bereits angedeutet, wird mit diesem Kriterium auch ignoriert, dass die Funktionen nicht isoliert, sondern stets im Kontext eines Softwaresystems beziehungsweise eines übergeordneten Supersystems zu betrachten sind. Das Verhalten einer betrachteten Funktion

entspricht gegebenenfalls nur dann der Vorgabe, wenn über die eigentlichen spezifizierten Eingaben hinaus zusätzliche Eingaben auf anderen Kanälen übertragen werden.

Überhaupt gibt es verschiedene Formen der Unabhängigkeit, die einer sorgfältigen Analyse bedürfen. Beispielsweise kann sich die Unabhängigkeit auf syntaktische Gegebenheiten beziehen. Wir sprechen von einer syntaktischen Disjunktheit, wenn die Eingabekanäle und die Ausgabekanäle der Funktionen disjunkt sind. In Abschnitt 4.2 wurde bereits aufgezeigt, dass sich jedoch rein aus der syntaktischen Disjunktheit heraus noch keine Rückschlüsse hinsichtlich des Einflusses auf das Verhalten einer Funktion ziehen lassen. Genauso wenig kann aus einer Verhaltensunabhängigkeit zwischen zwei Funktionen die syntaktische Konstellation ableitet werden.

Bevor wir die Bedeutung unabhängigen Verhaltens einer Funktion im Detail untersuchen, definieren wir zunächst die dafür in dieser Arbeit verwendete Notation.

Definition 4.10 (Der Beziehungstyp independent)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn in s die Funktion g in ihrem Verhalten unabhängig von f ist, so schreiben wir

$$\text{`}g\text{'}\ \textbf{\textit{independent\_of}}\ \text{`}f\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{independent}}(g, f)$$

□

Wie bereits erwähnt, ist die Unabhängigkeitsrelation nicht symmetrisch. Ist die betrachtete Funktion g unabhängig in ihrem Verhalten von f, folgt daraus nicht, dass f auch unabhängig in ihrem Verhalten von g ist. Sollte aber auch die Umkehrung zutreffen, bezeichnen wir den Zusammenhang der Funktionen als wechselseitig unabhängig.

Definition 4.11 (Wechselseitige Unabhängigkeit von Funktionen)

Zwei kombinierbare Funktionen f und g sind wechselseitig unabhängige Funktionen, wenn sowohl f in ihrem Verhalten unabhängig von g, als auch g in ihrem Verhalten unabhängig von f ist.

$$\textbf{\textit{independent}}(f, g) \ \wedge \ \textbf{\textit{independent}}(g, f)$$

□

Wechselseitig unabhängige Funktionen sind stets ohne Einschränkungen kombinierbar. Sie üben keinen Einfluss auf das Verhalten der anderen Funktion aus, ganz gleich ob sie nebenläufig oder nacheinander benutzt werden, denn ihre Eingaben können sich beliebig

überlappen. Was für das Systemdesign günstig erscheinen mag, wird andererseits aber oftmals gar nicht benötigt.

Beispielsweise sind die beiden Funktionen «Anruf annehmen» und «Nummer anrufen» eines Mobiltelefons nicht wechselseitig unabhängig. Es ist leicht nachvollziehbar, dass zwar «Anruf annehmen» in ihrem Verhalten unabhängig von «Nummer anrufen» ist. Die Funktion «Nummer anrufen» dagegen kann und soll durch «Anruf annehmen» abgebrochen werden.

Generelle Unabhängigkeit

Bevor wir speziell die Beziehung zwischen zwei Funktionen untersuchen, interessiert uns zunächst, ob sich allgemein Aussagen bezüglich der Unabhängigkeit einer betrachteten Funktion treffen lassen. So kann man etwa in der Voraussetzung für die unbeeinflusste Einbettung einer Funktion weder den Bezug zu einer speziellen Funktion noch zu einzelnen Eingabehistorien feststellen (siehe Seite 81).

Wenn g unbeeinflusst in s eingebettet ist, folgt daraus insbesondere für eine spezielle Funktion f in s, dass diese das Verhalten von g selbstverständlich auch nicht beeinflussen kann.

$$g \text{ ist unbeeinflusst eingebettet in } s \;\Rightarrow\; \text{`}g\text{'} \textbf{ independent\_of } \text{`}f\text{'} \textbf{ in } s$$

Diese Implikation ist offenkundig sehr allgemein. Falls nun aber g nicht unbeeinflusst in s eingebettet ist, kann g gleichwohl unabhängig von f sein?

Unabhängigkeit von einer speziellen Funktion

Dies ist zweifelsohne möglich, aber die Beschreibung der Verhaltensunabhängigkeit bezüglich einer speziellen Funktion f gestaltet sich gerade im Kontext eines multifunktionalen Supersystems als nicht trivial. Zunächst erscheint es nahe liegend von einer Unabhängigkeit zu sprechen, wenn in einer Eingabehistorie x gültige Eingaben von f vorkommen und diese nicht zu einem abweichenden Verhalten beitragen.

$$\forall\, x \in P^s_V(g) : x \in P^s_V(f) \;\Rightarrow\; x \in P^s_N(g)$$

Mit dieser Implikation trifft man allerdings nur den nicht vorhandenen Einfluss von Nachrichten über andere Eingabekanäle als I_g. Darüber hinaus muss die Ausgabe aber auch bei gemeinsamen Eingabekanälen vorgabegemäß sein und x somit auch ein Element von $P^s_S(g)$ sein.

Ferner ist die Bedingung, dass $x \in P^s_N(g)$ sein soll, zu stark. Denn mit ihr werden zwar Eingaben auf allen Eingabekanälen von s berücksichtigt, nicht aber die spezielle Situation für Eingaben nur auf $I_g \cup I_f$. Die Eingabe x kann ohne weiteres auch in der Menge der

beeinflussenden Eingaben von g liegen, $x \in P^s_A(g)$, während s sich bei der Restriktion der Eingaben auf die Kanäle $I_g \cup I_f$ neutral verhält.

$$\exists\, x \in P^s_A(g) : s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f)$$

Ist das Verhalten von g nun tatsächlich unabhängig von f, so wird aus Sicht der betrachteten Funktion g einerseits die Domäne von s sämtliche Überlagerungsmöglichkeiten aller gültigen Eingaben von g und f enthalten.

$$\forall\, x' \in \{x_g \oplus x_f : x_g \in \mathsf{dom}(g) \ \wedge \ x_f \in \mathsf{dom}(f)\} : \exists\, x \in \mathsf{dom}(s) \text{ mit } x' = (x \,|\, I_g \cup I_f)$$

Auf der anderen Seite müssen dann auch die auf O_g projizierten Ausgaben denen der Vorgabe in g entsprechen.

$$s\dagger(I_g \cup I_f \blacktriangleright O_g).x' = g.(x' \,|\, I_g)$$

Kriterien für die Unabhängigkeit – 'g' independent_of 'f'

Zusammenfassend können wir also von einer Verhaltensunabhängigkeit von g gegenüber f sprechen, wenn für alle $x \in P^s_V(g)$ gilt:

1. In x ist keine gültige Eingabe von f enthalten. In diesem Fall ist trivialerweise keine Abhängigkeit gegeben.

   $$x \notin P^s_V(f)$$

Ist dagegen in x eine gültige Eingabe von f enthalten, kann diese sich beliebig mit einer gültigen Eingabe von g überlappen, $(x \,|\, I_g \cup I_f) \in \{x_g \oplus x_f : x_g \in \mathsf{dom}(g) \ \wedge \ x_f \in \mathsf{dom}(f)\}$. Dann ist g unter den folgenden Prämissen unabhängig von f:

2. Die Ausgabe entspricht unabhängig von der Wahl der Eingabekanäle der Vorgabe. Auch hier ist trivialerweise keine Abhängigkeit gegeben.

   $$x \in (P^s_S(g) \cap P^s_N(g))$$

3. Die Ausgabe weicht ab, nur wenn neben den Eingaben auf I_g noch Eingaben auf zusätzlichen Kanälen berücksichtigt werden. Die Funktion f ist dafür aber nicht verantwortlich.

   $$x \in (P^s_S(g) \cap P^s_A(g)) \ \wedge \ s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f)$$

Auch wenn Eingaben über I_f mit in die Betrachtung einbezogen werden, hat dies keinen Effekt. Das Verhalten entspricht stets der Vorgabe in g. Es spielt dann hierbei auch keine Rolle, ob die Eingabekanäle von g und f disjunkt sind.

4. Die Ausgabe weicht zwar ab, auch wenn nur Eingaben auf I_g erfolgen. Die Funktion f ist dafür aber nicht verantwortlich.

   $$x \in (P_D^s(g) \cap P_N^s(g)) \ \wedge \ I_g \cap I_f = \varnothing$$

   Dies schließt insbesondere auch die Möglichkeit von widersprüchlich definierten Ausgaben aus.

   Theoretisch besteht auch die Möglichkeit, dass g unabhängig von f ist, obwohl beide Funktionen gemeinsame Eingabekanäle besitzen. Der Nachweis der Unabhängigkeit ist dann aber kaum möglich, denn das Ausblenden von Eingabekanälen ist an dieser Stelle ungeeignet. Beispielsweise kann etwa noch eine Funktion h in s existieren, die ebenso gemeinsame Eingabekanäle ($I_g \cap I_h \neq \varnothing$) aufweist und die letztlich die Ursache für das abweichende Verhalten ist.
   Auch ist es denkbar, dass im Rahmen der Festlegung des Gesamtverhaltens von s erst ein bestimmter Systemzustand gegeben sein muss, damit die Ausgaben wie spezifiziert eintreten. Wenn also gleichsam per Voreinstellung die Abweichung gegeben ist und g zum Beispiel standardmäßig deaktiviert ist, spielt dafür die Eingabe einer anderen Funktion in s und damit insbesondere auch der Funktion f keine Rolle.

   Das Aktivierungsszenario bringt uns zum letzten Diskussionspunkt hinsichtlich gültiger Eingaben von g.

5. Die Funktion f ist ebenso nicht dafür verantwortlich, dass nicht konformes Verhalten berichtigt wird.

   $$x \in (P_D^s(g) \cap P_A^s(g)) \ \wedge \ s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f)$$

   Weicht das Verhalten nur bei Eingaben über I_g bereits ab – und f darf dafür nicht die Ursache sein (siehe 4.) – wird auch die Nutzung der Funktion f nichts daran ändern. Man beachte, dass hier insbesondere die Ausgabemengen $(s.x) \mid O_g$ und $s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f)$ verschieden sind.

Soweit die Unabhängigkeitsbedingungen für Historien, welche gültige Eingaben von g beinhalten. Der Leser erinnere sich an dieser Stelle an die Menge $P_0^s(g)$ der undefinierten Eingaben von g in s.

Wenn eine Eingabehistorie $x \in P_0^s(g)$ zwar undefinierte Eingaben in g enthält, zugleich dort aber die Eingaben einer anderen Funktion f definiert sind, sprechen wir von einer

Eingabeerweiterung von g (extend), falls diese in Ausgaben aus dem Wertebereich von g resultieren. Mit anderen Worten: in der Kombination mit f führen an sich nicht definierte Eingaben zu bereits in g definierten Ausgaben. Auf das Verhalten hinsichtlich des definierten Definitionsbereichs und Wertebereichs in g hat f zwar keinen Einfluss. Dennoch kann ein Zusammenhang zwischen den Funktionen wahrgenommen werden, insbesondere weil sich die Wertebereiche der Funktionen überschneiden. Wir unterstellen daher auch für diesen Fall, dass g nicht unabhängig von f ist. Damit 'g' **independent_of** 'f' schließlich gegeben ist, fordern wir deshalb zusätzlich:

6. Wenn x keine in g definierte Eingabe enthält, darf für eine in f definierte Eingabe notwendigerweise keine Ausgabe aus dem Wertebereich von g an O_g zu beobachten sein. Für alle $x \in P_0^s(g)$ gilt

$$x \in P_V^s(f) \;\Rightarrow\; s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) \not\subseteq \mathsf{ran}(g)$$

Kapitel 5

Das Verhalten kombinierter Funktionen

Beziehungstypen charakterisieren die Ursachen für ein bestimmtes kombiniertes Verhalten von Funktionen. Eine konkrete Beziehung zwischen zwei Funktionen offenbart sich insbesondere im dynamischen Zusammenspiel der Funktionen als kombiniertes Schnittstellenverhalten. Die kombinierte Nutzung zweier Funktionen und das beobachtete Verhalten im Vergleich zu den Vorgaben der Spezifikationen lassen Rückschlüsse auf die Beziehung zwischen den Funktionen zu.

Die Situation stellt sich bei der Anforderungsanalyse genau umgekehrt dar. Die Festlegung einer Beziehung ist die Ursache und lässt ein bestimmtes Verhalten in Kombination, die Wirkung, erwarten. Um nun einer Ursache auch die gewünschte Wirkung folgen zu lassen, bedarf es im Hinblick auf Spezifikation und Entwurf geeigneter Konsequenzen und Maßnahmen, wie zum Beispiel der passenden Festlegung von Schnittstellen, der Einführung von Systemmodi, der Definition von Vor- und Nachbedingungen oder der Vorkehrungen für Unterbrechbarkeit. Wir verwenden daher zur Beschreibung von Beziehungstypen im Folgenden nicht nur die in den Kapiteln 3 und 4 eingeführten formalen Notationen, sondern auch adäquate Spezifikationsschemata. Die geeigneten Maßnahmen zur Forcierung einer Beziehung zwischen Funktionen hängen naturgemäß von der gewählten Spezifikationsumgebung ab. Wir zeigen daher beispielhaft mögliche Konsequenzen anhand von Zustandsübergangsdiagrammen auf.

In diesem Kapitel beschäftigen wir uns mit der Klasse von Beziehungen, die in Kapitel 4.1 als Verhaltensbeziehungen eingeordnet wurden. Zu Beginn des Abschnitts 4.4 wurde für derartige Zusammenhänge von Funktionen bereits das Attribut „horizontal“ gebraucht (vergleiche Seite 69). Zwei Funktionen f und g stehen hier also nicht selbst in einer Subfunktionsrelation,

$$\neg\mathbf{subfunction}(f, g) \ \wedge \ \neg\mathbf{subfunction}(g, f)$$

aber sie sind gegebenenfalls Subfunktionen eines gemeinsamen Supersystems. Ein Beziehungstyp charakterisiert hier den Einfluss einer Teilfunktion auf eine andere Teilfunktion im Kontext dieses gemeinsamen Supersystems. Der Spezialfall der trivialen Kombinierbarkeit von f und g wird daher bei den folgenden diskutierten Beziehungstypen ausgeschlossen.

Wenn die Nutzung einer Funktion Auswirkungen auf den Zugang für die andere Funktion hat, wird dies als bedingte Kombinierbarkeit der Funktionen beziehungsweise als eine eingeschränkte Subfunktionsbeziehung der abhängigen Funktion mit einer Superfunktion dokumentiert.

Nutzungssicht und architekturelle Beziehungen

An dieser Stelle sei noch einmal betont, dass architekturelle Zusammenhänge logischer Komponenten nicht relevant für die Diskussion der hier folgenden Funktionsbeziehungen sind. Wie bereits in Kapitel 4.1 erwähnt, spielt die Klasse der architekturellen Beziehungen nur eine untergeordnete Rolle. Letztendlich geht es zwar auch dabei um die Übertragung von Eingabe- und Ausgabenachrichten. Allerdings sind Beziehungen aus Architektursicht nicht mit denen aus Nutzungssicht zu verwechseln. Bei architekturellen Zusammenhängen steht die Komposition von Funktionen im Blickpunkt, die bereits syntaktische Anforderungen im Sinne interner Interaktionsschnittstellen nach sich ziehen. Nachrichten über diese interne Schnittstellen sind aus Nutzungssicht jedoch nicht zu beobachten und daher nicht unmittelbar relevant.

Selbstverständlich kann eine Funktion das Ergebnis einer anderen Nutzungsfunktion benötigen. Beispielsweise kann die Interaktion mit dem Nutzer vom Resultat einer Passwort- oder PIN-Eingabe abhängen. Wir betrachten aber auch das Abstützen einer Funktion auf eine andere Funktion nicht aus architektureller Sicht, sondern ausschließlich aus der Perspektive der Nutzung. Vor diesem Hintergrund bleiben dafür im Wesentlichen zwei Arten des Zusammenhangs zu untersuchen. Diese offenbaren zwar keine Wirkung hinsichtlich eines potenziell nicht vorgabegemäßen Verhaltens, doch auch das „Heranziehen“ beziehungsweise das „Auslösen“ einer anderen Funktion wird subjektiv als Abhängigkeit wahrgenommen.

Wir werden für derartige Nutzungsbeziehungen die Begriffe „consult“ und „trigger“ verwenden, um diese unmissverständlich von den Zusammenhängen aus Architektursicht zu differenzieren. Bezeichnungen wie etwa „call“ würden keinen geeigneten Zusammenhang zwischen Nutzungsfunktionen suggerieren, sondern bereits einen technischen Zusammenhang im Rahmen einer logischen Komponentenarchitektur umschreiben. Beispielsweise wird auch bei den CARTRONIC Kommunikationsbeziehungen [BSDV97] lediglich zwischen den Interaktionsformen „mit Antwort“ und „ohne Antwort“ oder zwischen „request-response“- und „one-way“-Beziehungen [Rit04] unterschieden.

Beeinflussung des Verhaltens aus Nutzungssicht

Die Diskussion in Abschnitt 4.6 hat gezeigt, dass eine Funktion zweier kombinierbarer Funktionen in ihrem Verhalten unabhängig von der anderen Funktion ist, wenn diese weder direkt (zum Beispiel bei gemeinsamen Ausgaben) noch indirekt (etwa durch Manipulation des Zustands der Superfunktion) von ihr beeinflusst wird.

Gibt es einen anderen Zusammenhang als die Unabhängigkeit zwischen zwei kombinierbaren Funktionen, bezeichnen wir diesen zunächst allgemein mit affect.

Definition 5.1 (Der Beziehungstyp affect)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn in s die Funktion g in ihrem Verhalten von f beeinflusst wird, so schreiben wir

$$\text{`}f\text{'}\ \textbf{\textit{affects}}\ \text{`}g\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{affect}}(f, g)$$

g ist in ihrem Verhalten nicht unabhängig von f.

$$\textbf{\textit{affect}}(f, g) \ \Rightarrow\ \neg\textbf{\textit{independent}}(g, f)$$

□

Die Art der Beeinflussung kann anhand zustandsabhängiger Eingaben und Ausgaben charakterisiert werden. In einem Zustandsübergangsdiagramm werden etwa zur Spezifikation einer Superfunktion, welche die beiden kombinierbaren Funktionen umfasst, Zustände zur Steuerung des Kontrollflusses eingeführt, die wiederum Einfluss auf die möglichen Eingaben und Ausgaben der Subfunktionen haben. Je nach Art des beobachtbaren Einflusses wird ein entsprechender Beziehungstyp formuliert.

Aus den Kriterien für die Unabhängigkeit (vergleiche Seite 96) leiten sich nun folgende Bedingungen ab, die den Einfluss einer Funktion f auf eine betrachtete Funktion g manifestieren.

- Es gibt eine Eingabe $x \in P_V^s(g)$, die eine gültige Eingabe von f enthält, $x \in P_V^s(f)$, und deshalb dafür verantwortlich ist, dass
  - die Ausgabe abweicht,
  - es bei gemeinsamen Eingaben widersprüchliche Ausgaben gibt,
  - oder nicht konformes Verhalten berichtigt wird.
- Oder es gibt eine Eingabe $x \in P_0^s(g)$, die eine gültige Eingabe von f enthält, $x \in P_V^s(f)$, und in einer Ausgabe aus dem Wertebereich von g resultiert.

Die Beziehungstypen im Detail

Im Rahmen der Diskussion von Verhaltensmustern sowie der Kombinierbarkeit von Funktionen war bereits von Funktionen die Rede, die sich aufgrund ihrer widersprüchlich spezifizierten Ausgaben ausschließen (siehe Abschnitt 4.5). Im unmittelbar anschließenden Abschnitt untersuchen wir die Voraussetzungen und Möglichkeiten, solche Funktionen dennoch in einem gemeinsamen Supersystem zu kombinieren.

Im darauf folgenden Abschnitt stehen weniger gemeinsame Eingaben, als allgemein die Überlappungsmöglichkeiten der Eingaben in Vordergrund. Wie bereits kurz bei der Diskussion der Kombination kombinierbarer Funktionen angesprochen, führen gegebenenfalls nicht beliebig überlappende Eingabehistorien zu dem in der betrachteten Funktion vorgegebenen Verhalten (vergleiche Seite 92). Oft reicht die alleinige Einhaltung der vorgegebenen Konventionen für die beabsichtigte Nutzung einer Funktion nicht aus. Wir gehen dann von einer Kontrolle des Zugangs zu einer Funktion aus, wenn es darüber hinaus bestimmte Eingabehistorien einer anderen Funktion zu beachten gilt.

Schließlich untersuchen wir Zusammenhänge zwischen Funktionen, die sich zwar nicht unmittelbar anhand abweichender Ein- oder Ausgabehistorien festmachen lassen, von den Nutzern der Funktionen dennoch als nicht unabhängig wahrgenommen werden. Die Unterbrechung einer Funktion oder die Erweiterung der Eingabemöglichkeiten sind hierfür typische Beispiele.

5.1 Nicht kombinierbare Funktionen

Warum sollten nicht kombinierbare Funktionen überhaupt kombiniert werden? Bevor diese Frage beantwortet wird, klären wir zunächst noch einmal die Bedingungen der Nichtkombinierbarkeit. Zunächst sind nicht kombinierbare Funktionen offensichtlich nicht unabhängig voneinander. Das heißt, es gibt mindestens jeweils eine Eingabehistorie der Funktionen g und f, die sich nicht beliebig überlagern können, ohne einen entsprechenden Einfluss auf das Verhalten von g in s beziehungsweise f in s beobachten zu können.

Dieser Einfluss begegnet uns nun bei nicht kombinierbaren Funktionen als Widerspruch der Ausgabeanforderungen. Für mindestens eine gemeinsam gültige Eingabehistorie definiert jede Funktion voneinander verschiedene Ausgabehistorien. Kombinierte man solche Funktionen dennoch, würde sich für eine bestimmte Eingabe die Menge der Ausgaben vergrößern. Sinnvollerweise trifft dies überhaupt nur dann zu, wenn es gemeinsam definierte Eingaben der Funktionen gibt. Um diese Situation zu beschreiben, verwenden wir den Beziehungstyp exclude.

Definition 5.2 (Der Beziehungstyp exclude)

Wenn die Vorgaben zweier Funktion f und g bezüglich ihrer Ausgaben im Widerspruch stehen, so schreiben wir

$$\text{`}f\text{'} \ \textbf{\textit{excludes}} \ \text{`}g\text{'} \quad \textit{oder} \quad \textbf{\textit{exclude}}(f, g)$$

Die Funktionen f und g besitzen gemeinsame Eingabe- und Ausgabekanäle sowie mindestens eine gemeinsame Eingabe, die aus der Perspektive von g zu zusätzlichen Ausgaben führt. □

Würden f und g in ein gemeinsames System s kombiniert werden, ließe sich für diesen Einfluss von f etwa folgende notwendige Bedingung formulieren.

$$\begin{aligned} &I_f \cap I_g \neq \varnothing \text{ und es existiert ein } x \in (P^s_D(g) \cap P^s_N(g)) : \\ &\quad x \in P^s_V(f) \ \wedge \ g.(x \,|\, I_g) \subset s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) \end{aligned}$$

Wenn eine Eingabe x ein Element von $P^s_D(g)$ und zugleich von $P^s_N(g)$ ist, kommen nur gemeinsame Eingabenachrichten für das nicht vorgabegemäße Verhalten in Frage. Zusätzlich ist x auch ein Element der Menge $P^s_V(f)$ der gültigen Eingaben für f. Damit wird sichergestellt, dass eine in f definierte Eingabehistorie auch in x enthalten ist. Schließlich entspricht die Ausgabe auf O_g nicht der Vorgabe in g, indem sich die Menge der Ausgabehistorien vergrößert.

Prinzipiell wären die Funktionen f und g also nicht kombinierbar. Eine Kombination der Funktionen in ein gemeinsames Softwaresystem aber nur aufgrund der zunächst widersprüchlich spezifizierten Ausgaben auszuschließen, entspricht nicht immer den Anforderungen. Beispielsweise sollte mit derselben Eingabe ein Mobiltelefon sowohl eingeschaltet als auch ausgeschaltet werden können (etwa durch langes Drücken der Auflegen-Taste). Die Kombination der Funktionen ist selbstverständlich möglich, weil sie niemals zugleich zugänglich sind.

Um nun widerspruchsbehaftete Funktionen dennoch zu kombinieren und dabei „unwanted feature interactions“ zu vermeiden, sind geeignete Maßnahmen erforderlich. Die einzige Möglichkeit, solche Funktionen konsistent zu kombinieren, besteht darin, die simultane Nutzung zu verhindern.

Verschiedene Betriebsmodi eines Softwaresystems

Dies kann zum Beispiel durch die Einführung verschiedener Systemzustände eines übergeordneten Supersystems s realisiert werden, in denen jeweils nur eine der Funktionen genutzt werden kann. Die Funktionen sind dann *bedingt kombinierbar* unter der Bedingung, dass der Zugang zu f und der Zugang zu g jeweils einen anderen Betriebsmodus erfordert. Die Eingaben der Funktionen überlappen sich dann niemals in unerwünschter

Art und Weise. Die Menge der Eingabehistorien von s, bei denen sich die Eingaben von g und f nicht überlappen, sei mit $P^s_{EXC}(g,f)$ bezeichnet[1].

$$P^s_{EXC}(g,f) = \left\{ x \in P^s_V(g) : x \in P^s_V(f) \;\Rightarrow\; \begin{array}{ll} & \mathsf{ltm}(x \,|\, I_g) < \mathsf{ftm}(x \,|\, I_f) \\ \vee & \mathsf{ltm}(x \,|\, I_f) < \mathsf{ftm}(x \,|\, I_g) \end{array} \right\}$$

Falls eine Eingabehistorie eine gültige Eingabe von f enthält, wird ihre erste Nachricht nach der letzten Nachricht von g übertragen oder ihre letzte Nachricht vor der ersten Nachricht von g.

Aus der Perspektive der betrachteten Funktion g lässt sich die oben genannte Exklusivitätsbeziehung nun abkürzend auch mit

$$\textbf{subfunction}(s \,|\, P^s_{EXC}(g,f), g)$$

formulieren. Diese Schreibweise soll andeuten, dass g nur dann eine Teilfunktion von s ist, wenn f nicht zugleich genutzt werden kann.

Über den Gebrauch verschiedener Betriebsmodi hinaus kann noch ein weitergehender Zusammenhang der Funktionen existieren. Eine Funktion kann gewissermaßen auch ursächlich dafür verantwortlich sein, überhaupt in einen anderen Betriebszustand zu gelangen. Am Beispiel des Mobiltelefons oben kann man sich vorstellen, dass mit der Funktion «Einschalten» etwa ein Zustand *Standby* erreicht werden kann, welcher wiederum erst die Voraussetzung für den Zugang zur Funktion «Ausschalten» darstellt. Wir bringen einen derartigen Zusammenhang zwischen Funktionen mit dem Beziehungstyp prepare zum Ausdruck.

Auch eine prepare Beziehung ist zweifelsohne dafür geeignet, Ausgabekonflikte zu lösen. Die Nutzung von Funktionen in unterschiedlichen Betriebsmodi steht aber nicht notwendigerweise im Zusammenhang mit Maßnahmen zur konsistenten Kombination von Funktionen. Das heißt, es kann auch für an sich unbedingt kombinierbare Funktionen die Anforderung geben, dass

(a) sie nicht in demselben Betriebsmodus genutzt werden sollen und
(b) eine der Funktionen sogar erst in den für die andere Funktion notwendigen Systemzustand führt.

[1] $\mathsf{ftm}(x)$ und $\mathsf{ltm}(x)$ geben das Zeitfenster der ersten beziehungsweise letzten übertragenen Nachricht in x an, vergleiche Definition 3.6.

Definition 5.3 (Der Beziehungstyp prepare)

Zwei Funktionen f und g seien eingebettet in ein Supersystem s[2]. Wenn erst mit der Nutzung von f ein Betriebszustand von s erreicht wird, welcher den Zugang zu g ermöglicht, so schreiben wir

$$\text{`}f\text{'}\ \textbf{\textit{prepares}}\ \text{`}g\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{prepare}}(f, g)$$

□

Im Übrigen sind auch wechselseitige prepare Beziehungen nicht unüblich. Mit dem «Ausschalten» eines Mobiltelefons gelangt man beispielsweise erst in den Systemmodus, der das «Einschalten» erst wieder zulässt.

Doch was bedeutet dies nun für Funktionen, die in solch einer Beziehung stehen? Zunächst existiert dann und nur dann ein Zugang zu der betrachteten Funktion g, nachdem eine in f definierte Interaktion erfolgt ist. Darüber hinaus ist bis auf weiteres die Nutzung von f ausgeschlossen. Mit anderen Worten: die Nutzung von f ebnet zum einen die Nutzung von g und schließt zum anderen ihre eigene wiederholte Nutzung aus.

Bei einer prepare Beziehung wird also die abhängige Funktion nicht im Sinne einer Aktivierung beeinflusst, sondern ist vielmehr dahingehend zu sehen, als dadurch ein geeigneter Zustand des Systems erreicht wird. Konsequenterweise legen prepare Beziehungen zwischen Funktionen die Einführung geeigneter Betriebsmodi nahe, für deren Erreichung die Einfluss nehmenden Funktionen zwingend erforderlich sind. Bei der Verwendung von Zustandsübergangsdiagrammen können gegebenenfalls die Endzustände einer Funktion sowie der Startzustand der abhängigen Funktion als gemeinsame Kontrollzustände zusammengefasst werden.

Prioritäten von Funktionen

Eine andere Möglichkeit, die simultane Nutzung widerspruchsbehafteter Funktionen zu umgehen, ist die bewusste Ersetzung der Ausgabe einer Funktion durch die der anderen Funktion.

Im Allgemeinen werden die Ausgabenachrichten in f nicht in g definiert sein, beziehungsweise werden die Nachrichten sogar auf anderen Kanälen als O_g ausgegeben. Die Projektion auf die Ausgabekanäle von g führt dann zu leeren Nachrichtenströmen, präziser ausgedrückt: zu Strömen, in denen keine Nachrichten übertragen werden. An dieser Stelle sei darauf hingewiesen, dass auch ein leerer Nachrichtenstrom als zusätzliche Ausgabe zur Vergrößerung der Ausgabemenge beiträgt.

[2] Ob die Funktionen kombinierbar sind oder nicht ist hier nicht wesentlich, da aufgrund der «prepare» Beziehung die Funktionen nur in unterschiedlichen Betriebsmodi genutzt werden können und damit potenziell widersprüchliche Ausgaben nicht zum Tragen kommen.

Angenommen es gibt nun unterschiedliche Ausgaben der kombinierten Funktionen. Wenn der Einfluss nehmenden Funktion eine höhere Priorität beigemessen wird und damit lediglich deren Ausgabe zum Tragen kommt, wird dies durch den Beziehungstyp replace ausgedrückt.

Definition 5.4 (Der Beziehungstyp replace)

Zwei Funktionen f und g seien eingebettet in ein Supersystem s. Wenn es eine gemeinsame Eingabe von g und f gibt, für die jeweils die Ausgabe unterschiedlich spezifiziert ist, in s aber nur die in f definierte Ausgabe zu beobachten ist, so schreiben wir

*'f' **replaces** 'g' **in** s oder* $\mathbf{replace}(f, g)$

□

Eine notwendige Bedingung lässt sich analog zum exclude Beziehungstyp formulieren, nur dass die auf O_g projizierte Ausgabemenge eine Historie beinhaltet, welche nicht in g spezifiziert ist[3].

$$I_f \cap I_g \neq \varnothing \text{ und es existiert ein } x \in (P^s_D(g) \cap P^s_N(g)) :$$
$$x \in P^s_V(f) \ \wedge \ \exists\, y \in s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) : y \notin g.(x \,|\, I_g)$$

Mit der Priorisierung einer der Funktionen kann nicht eindeutiges Verhalten an der Schnittstelle verhindert werden. Allerdings wird die Ausgabe einer Funktion durch die Ausgabe der anderen Funktion ersetzt. Mit 'f' **replaces** 'g' wird zum Ausdruck gebracht, dass die Ausgabemenge für eine bestimmte Eingabe auf die Ausgaben von f reduziert wird.

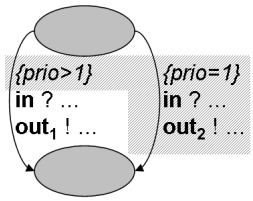


Abbildung 5.1: Schema einer replace Beziehung im Zustandsübergangsdiagramm

In dem Schema für Zustandsübergangsdiagramme in Abbildung 5.1 soll mit der Kanalbezeichnung **in** angedeutet werden, dass es Transitionen mit gleichen Eingabenachrichten

[3] Im Falle eindeutiger Ausgaben, das heißt falls die Ausgabemengen jeweils nur ein Element beinhalten, wird eine Ausgabehistorie durch die andere Ausgabehistorie ersetzt.

gibt. Der unterlegte Bereich kennzeichnet nun die Verantwortlichkeit der Einfluss nehmenden Funktion. Mit der Festlegung entsprechender Vorbedingungen *{prio . . . }* können widersprüchliche Ausgaben verhindert werden.

Verfeinerung des Verhaltens einer Funktion

Wenn für eine bestimmte Eingabe die Ausgabemenge mehr als eine Historie enthält, so kann dies als nicht deterministisches Verhalten einer Funktion interpretiert werden. Mit einer replace Beziehung zwischen Funktionen können die durch nicht konsistent spezifiziertes Verhalten verursachten zusätzlichen Ausgaben nun umgangen werden. Insofern verursachte die Einfluss nehmende Funktion in einer replace Beziehung zwar zunächst die Vergrößerung der Ausgabemenge, zugleich wird diese aber wieder dank der Priorisierung reduziert.

Generell verwenden wir für die Reduzierung der Ausgabevielfalt einer Funktion den Begriff der Verhaltensverfeinerung. Wenn die Einfluss nehmende Funktion das Verhalten einer anderen Funktion bestimmt, indem sie für eine gegebene Eingabehistorie nur eine Teilmenge der Ausgabehistorien zulässt, dann drücken wir diesen Zusammenhang als refine Beziehung aus. Das Verhalten der betrachteten Funktion wird durch die andere eingebettete Funktion gewissermaßen präzisiert.

Da bei diesem Beziehungstyp insbesondere die Eliminierung nicht deterministischen Verhaltens eine Rolle spielt, ist die Bezeichnung refine nicht erstaunlich. Die Assoziation mit den Definitionen 3.15 und 4.1 der Verhaltensverfeinerung und des subfunction Beziehungstyps ist durchaus gewollt. Wenn man so will, kann man mit refine Beziehungen auch solche Funktionen identifizieren, welche für die Verhaltensverfeinerungen an den Schnittstellen der abhängigen Funktionen verantwortlich sind.

Definition 5.5 (Der Beziehungstyp refine)

Zwei Funktionen f und g seien eingebettet in ein Supersystem s. Wenn zu einer Eingabe für g verschiedene Ausgaben definiert sind, diese aber jeweils nur in Abhängigkeit von entsprechenden Eingaben für f zu beobachten sind, so schreiben wir

$$\text{`}f\text{'}\ \textbf{\textit{refines}}\ \text{`}g\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{refine}}(f, g)$$

□

Als notwendige Bedingung dafür, dass f in einer refine Beziehung mit g steht, muss demgemäß ein $x \in P^s_V(g)$ existieren mit

$$x \in P^s_V(f) \ \wedge \ s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) \subset g.(x \,|\, I_g)$$

Auch hier muss in x eine in f definierte Eingabehistorie enthalten sein. Die Ausgabe auf O_g entspricht nicht der Vorgabe in g, wobei sich die Menge der Ausgabehistorien verringert. Aus welchem Grund das Verhalten einer betrachtete Funktion unterbestimmt ist, sei es durch widersprüchliche Ausgaben von Funktionen oder sei es, dass in der Spezifikation einer betrachteten Funktion bereits nicht deterministisches Verhalten vorgesehen ist, ist für eine refine Beziehung unerheblich.

Die Reduzierung der Ausgabevielfalt kann man sich hier sowohl aufgrund von Eingaben auf gemeinsamen Kanälen von g und f vorstellen, oder aber über die Einbeziehung von Eingaben auf zusätzlichen Kanälen als I_g. Die Präzisierung der Ausgaben bei einer gemeinsamen Eingabe ist etwa im Rahmen der Integration in ein Supersystem s denkbar, wenn der Zustand von s derart manipuliert wird, dass die Ausgaben der betrachteten Funktion g unterdrückt werden. Dies kommt im Prinzip einer replace Beziehung gleich, bei der die Ausgabemenge der Einfluss nehmenden Funktion einer Teilmenge entspricht.

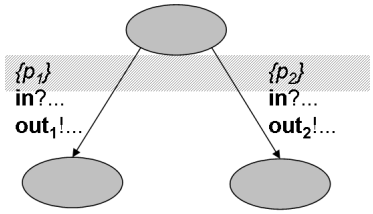


Abbildung 5.2: Schema einer refine Beziehung im Zustandsübergangsdiagramm

In Zustandsübergangsdiagrammen wird nicht deterministisches Verhalten, das heißt es gibt Transitionen mit gleichen Eingaben, durch Einführung geeigneter Vorbedingungen eliminiert. Die beeinflussende Funktion ist dann für die Erfüllung der Vorbedingungen durch das Setzen entsprechender Nachbedingungen verantwortlich. In Abbildung 5.2 ist ein solches Schema dargestellt. Eine Funktion, welche in einer refine Beziehung zu einer anderen Funktion steht, reduziert den Umfang derer möglichen Ausgabevarianten. Für alle Transitionsvorbedingungen $i \neq j$ muss mindestens ein p_i erfüllt und mindestens ein p_j nicht erfüllt sein.

5.2 Kontrollierter Zugang zu Funktionen

In der Kombination von Funktionen stellt sich unvermeidlich die Frage, wie man einzelne Funktionen eigentlich nutzen kann. Oft reicht es nicht aus, sich nur an den Nutzungskonventionen einer Funktion zu orientieren, also insbesondere sich an die in der Domäne einer Funktion definierten Eingaben zu halten. Vorher ist zu klären, in welcher Form der Zugang zu einer Funktion überhaupt gegeben ist. Der uneingeschränkte Zugang zu einer

Funktion geht allerdings nicht notwendigerweise mit der Unabhängigkeit einer Funktion einher. Abgesehen von dem Zugang zur Funktion können etwa Beziehungen zu anderen Funktionen existieren, welche Einfluss auf das Verhalten haben.

Wie wir im vorhergehenden Abschnitt gesehen haben, muss der Zugang zu einer Funktion in bestimmten Situationen sogar verhindert werden. Auf diese Weise können an sich nicht kombinierbare Funktionen dennoch mit gewisser Einschränkung kombiniert werden. Wenn der Zugang zu einer Funktion nun nicht in jedem Fall gegeben ist, stellt sich die Frage, ob es eine weitere Funktion gibt, die über die eingehaltenen Konventionen hinaus notwendig ist, um diese Funktion nutzen zu können. Beziehungsweise ob der Zugang zu einer Funktion gerade durch die Kombination mit einer anderen Funktion eingeschränkt wird.

Fraglos ist in solchen Kombinationen zumindest eine der Funktionen nicht unabhängig von der anderen Funktion. Wenn eine Funktion nur unter Beachtung bestimmter Eingabehistorien einer anderen Funktion in der spezifizierten Art und Weise genutzt werden kann, sprechen wir auch dann von einer bedingten Kombinierbarkeit der Funktionen.

Ermöglichung des Zugangs zu einer Funktion

Derartige zusätzlich benötigte Funktionen kontrollieren gleichsam den Zugang zu Funktionen, indem sie die Aufgabe des Aktivierens übernehmen. Es liegt auf der Hand, dass sich nur in der Kombination der Funktionen die betrachtete Funktion wie spezifiziert verhält. Bestimmte zusätzliche Eingaben der kontrollierenden Funktion sind unbedingt erforderlich. Wir bezeichnen die Menge dieser Eingabehistorien daher auch als *Enable Set*.

Definition 5.6 (P_{ENA}– Enable Set)

Die Menge aller Eingaben in s, bei denen die Ausgaben nur dann der Vorgabe in g entsprechen, wenn Nachrichten auf weiteren Kanälen als I_g berücksichtigt werden, wird mit $P^s_{ENA}(g)$ bezeichnet.

$$P^s_{ENA}(g) = \{x \in P^s_D(g) \cap P^s_A(g) : (s.x) \,|\, O_g = g.(x \,|\, I_g)\}$$

□

Nur wie in g spezifizierte Eingaben reichen nicht aus, damit die Ausgaben auch der Vorgabe in g entsprechen ($x \in P^s_D(g)$, also $g.(x \mid I_g) \neq s \dagger (I_g \blacktriangleright O_g).(x \mid I_g)$, vergleiche Definition 4.5). Dafür bedarf es zusätzlich der Nutzung einer anderen eingebetteten Funktion mit entsprechenden Eingaben auf weiteren Kanälen ($x \in P^s_A(g)$, also $s \dagger (I_g \blacktriangleright O_g).(x \mid I_g) \neq (s.x) \mid O_g$), vergleiche Definition 4.7), damit die Ausgabe wieder spezifikationskonform ist, das heißt es gilt $(s.x) \,|\, O_g = g.(x \,|\, I_g)$.

Die Menge der Kanäle aller beeinflussenden Eingaben von g entspricht nicht I_g. Da $I_A(g) \neq I_g$ gilt, gibt es mindestens eine von g verschiedene in s eingebettete Funktion, deren Eingaben auf den von I_g verschiedenen Kanälen erst zu einem spezifikationskonformen Verhalten von g in s führen. Was bedeutet dies nun im Hinblick auf die Kombinierbarkeit?

Die abhängige Funktion ist mit der geeigneten Nutzung einer sie kontrollierenden Funktion verbunden. Infolgedessen existierte ohne die kontrollierende Funktion f kein Zugang zu der abhängigen Funktion g. Die Kombinierbarkeit der Funktion muss darum an die Bedingung, die durch den Enable Set formuliert wird, geknüpft werden. Die Funktionen sind deshalb *bedingt kombinierbar* unter der Bedingung $P^s_{ENA}(g)$, da die Funktion g nur für alle Eingabehistorien aus $P^s_{ENA}(g)$ eine Subfunktion von s ist. Wir formulieren diesen Zusammenhang abkürzend auch mit

$$\textbf{subfunction}(s \,|\, P^s_{ENA}(g), g)$$

Definition 5.7 (Der Beziehungstyp enable)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn erst mit der geeigneten Nutzung von f der Zugang zu g gegeben ist, das Verhalten von g also nur in Zusammenhang mit gültigen Eingabehistorien von f konform zu der Vorgabe in g ist, so schreiben wir

$$\textit{`}f\textit{'} \ \textbf{\textit{enables}} \ \textit{`}g\textit{'} \ \textbf{\textit{in}} \ s \quad \textit{oder} \quad \textbf{\textit{enable}}(f, g)$$

□

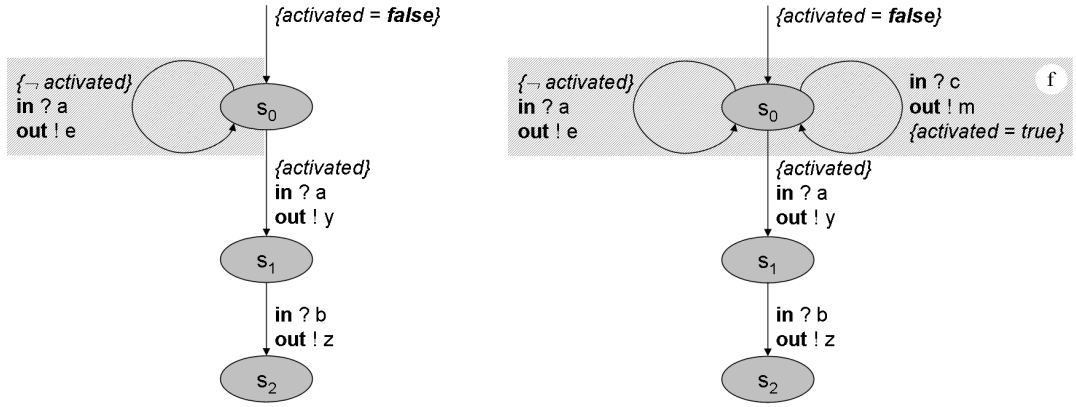
Als notwendige Bedingung dafür, dass f in einer enable Beziehung mit g steht, muss demgemäß ein $x \in P^s_{ENA}(g)$ existieren mit

$$x \in P^s_V(f) \ \wedge \ s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) = g.(x \,|\, I_g)$$

Eine in x enthaltene in f definierte Eingabehistorie ist dafür verantwortlich, dass die Ausgabe auf O_g der Vorgabe in g entspricht.

Üblicherweise wird beim Entwurf eines Systems ein Standardverhalten festgelegt, wenn eine Funktion nicht genutzt werden kann. Dies sind zwar dann Ausgaben außerhalb des Wertebereichs der beeinflussten Funktion, zum Beispiel eine Meldung der fehlerhaften Situation, die Domäne bleibt davon aber unberührt. Insofern schließen wir die in dem Fall theoretische Möglichkeit einer undefinierten Eingabe, $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g) = \varnothing$, aus.

In Zustandsübergangsdiagrammen lässt sich beispielsweise ein enable Zusammenhang über die Belegung von Zustandsvariablen erreichen. So ist etwa die in Abbildung 5.3(a)



(a) Voreingestellte Deaktivierung der Funktion g

(b) Integrierte, kontrollierende Funktion f

Abbildung 5.3: Schema einer enable Beziehung im Zustandsübergangsdiagramm

spezifizierte Funktion g standardmäßig deaktiviert, das an der mit *false* vorbelegten Zustandsvariablen *activated* zu erkennen ist. Aus diesem Diagramm wird im Übrigen auch ersichtlich, dass die Eingaben einer deaktivierten Funktion nicht notwendigerweise undefiniert sein müssen, sondern eben nur zu nicht vorgabegemäßen Ausgaben führen.

Die kontrollierende Funktion f weist nun im Rahmen einer Nachbedingung einer Variablen einen bestimmten Wert zu, hier *{activated=true}* in Abbildung 5.3(b). Dann verhält sich g in der spezifizierten Art und Weise.

Beschränkung des Zugangs zu einer Funktion

In der Kombination der Funktionen kann der Zugang zu einer kontrollierten Funktion nicht nur ermöglicht, sondern insbesondere auch unterbunden werden. Eine kontrollierte Funktion kann, solange die Nutzungskonventionen eingehalten werden, gemäß der Spezifikation genutzt werden. Allerdings können Eingaben einer anderen Funktion dazu führen, dass die Funktion nicht mehr genutzt werden kann. Auch hierbei kontrolliert eine Funktion den Zugang zu einer anderen Funktion, allerdings in der Form des Deaktivierens.

Betrachtet man nur die Kombination zweier Funktionen, so ist ohne die kontrollierende Funktion der Zugang zu der abhängigen Funktion stets gegeben. Bestimmte zusätzliche Eingaben der kontrollierenden Funktion verhindern jedoch die Nutzung der Funktion gemäß der Vorgabe. Die Menge dieser Eingabehistorien wird daher entsprechend als *Disable Set* bezeichnet.

Definition 5.8 (P_{DIS}– Disable Set)

Die Menge aller Eingaben in s, bei denen die Ausgaben nur dann von der Vorgabe in g abweichen, wenn Nachrichten auf weiteren Kanälen als I_g berücksichtigt werden, wird mit $P^s_{DIS}(g)$ bezeichnet.

$$P^s_{DIS}(g) = \{x \in P^s_S(g) \cap P^s_A(g) : (s.x) \mid O_g \not\subseteq g.(x \mid I_g) \ \wedge \ g.(x \mid I_g) \not\subseteq (s.x) \mid O_g\}$$

□

Die in g spezifizierten Eingaben führen zu vorgabegemäßen Ausgaben ($x \in P^s_S(g)$, also $g.(x \mid I_g) = s \dagger (I_g \blacktriangleright O_g).(x \mid I_g)$, vergleiche Definition 4.4). Die Ausgaben sind nicht mehr spezifikationskonform, wenn eine andere eingebettete Funktion mit entsprechenden Eingaben auf weiteren Kanälen genutzt wird ($x \in P^s_A(g)$, also $s\dagger(I_g \blacktriangleright O_g).(x \mid I_g) \neq (s.x) \mid O_g$), vergleiche Definition 4.7).
Man beachte, dass wir hier nicht nur die Ungleichheit der Ausgabemengen, $(s.x) \mid O_g \neq g.(x \mid I_g)$), fordern, da dies ist ein zu schwaches Kriterium ist. Wir schließen insbesondere die Teilmengenrelation aus, da widersprüchliche Ausgabemengen beziehungsweise die Reduzierung der Ausgabemenge im Kontext anderer Beziehungstypen relevant sind.

Analog zu einer enable Beziehung gilt auch hier $I_A(g) \neq I_g$. Es gibt mindestens eine von g verschiedene in s eingebettete Funktion, deren Eingaben auf den von I_g verschiedenen Kanälen erst zu einem von der Vorgabe in g abweichenden Verhalten führen.

Im Hinblick auf die Kombinierbarkeit der abhängigen Funktion mit anderen Funktionen ergibt sich das entsprechend inverse Bild. Die Funktionen sind *bedingt kombinierbar* für Eingaben, die nicht im *Disable Set* der kontrollierenden Funktion liegen. Für Eingabehistorien aus $P^s_{DIS}(g)$ ist die Funktion g keine Subfunktion von s. Oder positiv ausgedrückt: g ist eine Subfunktion von s für Eingabehistorien aus $\mathsf{dom}(s) \backslash P^s_{DIS}(g)$. Wir formulieren diesen Zusammenhang abkürzend auch mit

$$\textbf{subfunction}(s \setminus P^s_{DIS}(g), g)$$

Definition 5.9 (Der Beziehungstyp disable)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn mit der geeigneten Nutzung von f der Zugang zu g verhindert wird, das Verhalten von g also nur in Zusammenhang ohne bestimmte Eingabehistorien von f konform zu der Vorgabe in g ist, so schreiben wir

*'f' **disables** 'g' **in** s oder* $\textbf{disable}(f, g)$

□

Als notwendige Bedingung für eine disable Beziehung zwischen f und g muss analog zu oben ein $x \in P^s_{DIS}(g)$ existieren mit

$$\begin{array}{lcll} x \in P^s_V(f) & \wedge & s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) & \notin & g.(x \,|\, I_g) \\ & \wedge & g.(x \,|\, I_g) & \notin & s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) \end{array}$$

Eine in x enthaltene in f definierte Eingabehistorie ist dafür verantwortlich, dass die Ausgabe auf O_g nicht der Vorgabe in g entspricht. An dieser Stelle sei darauf hingewiesen, dass $P^s_{DIS}(g)$ im Allgemeinen kein Komplement von $P^s_{ENA}(g)$ ist. Intuitiv ist es auch nicht dasselbe, ob das (aktive) Einschalten einer Funktion mit Eingaben beeinflusst wird oder ob sie nur (passiv) ohne Eingaben nicht ausgeschaltet wird. Auch umgekehrt kann das Nicht-Einschalten einer Funktion mitnichten bereits mit der Deaktivierung derselben gleichgesetzt werden.

In Zustandsübergangsdiagrammen wird, wie beim enable Beziehungstyp bereits gezeigt, auch bei einer disable Beziehung die kontrollierende Funktion entsprechende Belegungen von Zustandsvariablen vornehmen – allerdings mit umgekehrten Vorzeichen. In Abbildung 5.3 wird etwa die Variable *activated* standardmäßig mit *true* belegt, während f sie auf *false* setzt.

Eine disable Beziehung kann man sich zum Beispiel zwischen den Funktionen «Nummer anrufen» und «Tasten sperren» eines Mobiltelefons vorstellen. Mit der Aktivierung der Tastensperre wird die unbeabsichtigte Nutzung des Geräts verhindert und damit insbesondere auch die Eingabe von Ziffern beim Wählen. Selbst wenn man die spezifizierte Interaktion zum Anrufen einer Nummer einhält, wird die Nutzung der Funktion «Nummer anrufen» unterbunden.

Die Bezeichnungen „enable“ und „disable“

Die Namen für die Beziehungstypen enable beziehungsweise disable sind bewusst gewählt. Einerseits sollen die mit diesen Begriffen charakterisierten Funktionsbeziehungen intuitiv einsichtig sein, andererseits aber auch Verwechslungen mit der feineren Bedeutung der Aktivierung oder der Deaktivierung von Systemzuständen vermieden werden [Grü08]. Dies spiegelt sich auch in den jeweiligen formalen Beschreibungen entsprechend wider, in denen lediglich Mengen von Eingaben und Ausgaben vorkommen.

Einfach ausgedrückt, sind bei einer enable Beziehung zwischen Funktionen bestimmte Eingaben der einen Funktion zwingend notwendig, damit sich die andere Funktion vorgabegemäß verhält. Während bei einer disable Beziehung bestimmte Eingaben der einen Funktion dazu führen, dass sich die andere Funktion eben nicht vorgabegemäß verhält. Wir unterstellen auch, dass sich der Definitionsbereich einer Funktion nicht ändert, auch wenn sie „disabled“ beziehungsweise „nicht enabled“ sein sollte. Im Allgemeinen sind also Ausgaben definiert, das heißt die Ausgabemenge ist nicht leer, nur entsprechen sie nicht der Vorgabe.

Mit solchen Funktionsbeziehungen wird jedoch nichts darüber ausgesagt, wie dieses Verhalten letzten Endes erreicht wird. Beispielsweise kann eine Funktion selbstverständlich für die Aktivierung eines Systemzustands verantwortlich sein, der wiederum dazu führt, dass sich eine andere Funktion spezifikationsgemäß verhält. Alternativ können aber auch geeignete Nachrichten über eine interne Interaktionsschnittstelle zwischen den Funktionen übertragen werden.

Selektive Interaktionssteuerung

Kommen wir nun zu dem letzten Beziehungstyp, welcher unter dem Begriff der „Zugangskontrolle“ anzusiedeln ist. Hier bestimmt die Einfluss nehmende Funktion die aktuell nutzbaren Eingabeoptionen, denn nicht alle definierten Eingaben sollen zu jeder Zeit zur Verfügung stehen. Die steuernde Funktion nimmt gezielt Einfluss auf die möglichen Kontrollflüsse der abhängigen Funktion und selektiert damit gleichsam deren Eingabeoptionen.

Definition 5.10 (Der Beziehungstyp direct)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn in g verschiedene Eingabeoptionen definiert sind, die allerdings nur in Abhängigkeit entsprechender Eingaben von f zugänglich sind, so schreiben wir

$$\text{‘}f\text{’ } \textbf{\textit{directs}} \text{ ‘}g\text{’ } \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{direct}}(f, g)$$

□

Ein und dieselbe Eingabe von g kann im Zusammenhang mit einer Eingabe von f in einer vorgabegemäßen Ausgabe resultieren, während die Ausgabe mit einer anderen Eingabe von f nicht der Spezifikation in g entspricht. Betrachten wir dazu eine Eingabe $x_g \in \mathsf{dom}(g)$, sowie zwei verschiedene in f definierte Eingaben x_{DIR}, $x'_{DIR} \in \mathsf{dom}(f)$. Zwei gültige Eingaben $x_1, x_2 \in P^s_V(g)$ enthielten jeweils x_g und die in f spezifizierten Eingaben x_{DIR} und x'_{DIR}.

$$\begin{array}{rlcl} & (x_1 \,|\, I_f) = x_{DIR} & \wedge & (x_1 \,|\, I_g) = x_g \\ \text{sowie} & (x_2 \,|\, I_f) = x'_{DIR} & \wedge & (x_2 \,|\, I_g) = x_g \end{array}$$

Wenn f in einer direct Beziehung mit g steht, dann gilt für x_1 und x_2

$$\begin{array}{rlcl} & s\dagger(I_g \cup I_f \blacktriangleright O_g).(x_1 \,|\, I_g \cup I_f) & = & g.(x_1 \,|\, I_g) \\ \wedge & s\dagger(I_g \cup I_f \blacktriangleright O_g).(x_2 \,|\, I_g \cup I_f) & \neq & g.(x_2 \,|\, I_g) \end{array}$$

In der Kombination der Funktionen steht stets nur ein Teil der in g spezifizierten Eingaben zur Verfügung, wobei der Umfang der Möglichkeiten durch Eingaben von f festgelegt

wird. Die Selektion der Eingabehistorien kann in einem Zustandsübergangsdiagramm beispielsweise gemäß des Schemas in Abbildung 5.4 umgesetzt werden. Der Teil, welcher in der Verantwortung der beeinflussenden Funktion liegt, ist auch hier rechteckförmig unterlegt. Wenn für eine Funktion verschiedene Eingabemöglichkeiten vorgesehen sind, so liegt der Einfluss der anderen Funktion nun darin, für die Erfüllung beziehungsweise Nicht-Erfüllung der Transitionsvorbedingungen zu sorgen. Bei einer direct Beziehung müssen für alle $i \neq j$ mindestens ein p_i erfüllt und mindestens ein p_j nicht erfüllt sein.

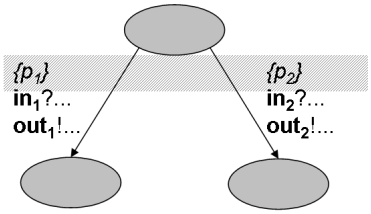


Abbildung 5.4: Schema einer direct Beziehung im Zustandsübergangsdiagramm

Hält man sich den Zugang zu einer Funktion vor Augen, ist eine gewisse Ähnlichkeit zu den Beziehungstypen enable beziehungsweise disable nicht zu verkennen. Dort steht allerdings die alleinige Steuerung des Zugangs im Vordergrund, während beim Beziehungstyp direct eine Selektion der Eingabehistorien ausschlaggebend ist. Dennoch kann man sich als Spezialfall eine direct Beziehung zwischen zwei Funktionen auch vorstellen als das parallele Aktivieren („enable“) und Deaktivieren („disable“) von Teilfunktionen, sofern es Zusammenhänge zwischen Funktionen gibt, wie sie in Abbildung 5.5 dargestellt sind.

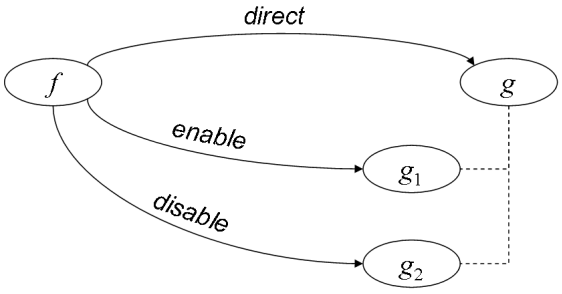


Abbildung 5.5: Spezialfall einer direct Beziehung

Kontrolliert eine Funktion f den Zugang zu den Subfunktionen g_1 und g_2 einer Funktion g, indem stets nur eine Subfunktion aktiviert und die andere Subfunktion deaktiviert wird, kann dies auch als eine direct Beziehung zwischen den Funktionen f und g aufgefasst werden.

5.3 Varianten und Muster von Einflussnahmen

Wir konzentrieren uns in dieser Arbeit bei der formalen Beschreibung von Funktionsbeziehungen rein auf die Beobachtung von Eingabe- und Ausgabehistorien mit geeigneten Restriktionen der Schnittstellen. Gerade bei den nun folgenden diskutierten Zusammenhängen stößt man aber mit diesen Mitteln an die Grenze, da sich im Allgemeinen an der Projektion der Eingaben und Ausgaben einer übergeordneten Superfunktion s auf die Schnittstelle der betrachteten Funktion g keine Beeinflussung erkennen lässt. Insbesondere steht formal die Kombinierbarkeit dieser Funktionen außer Frage. Eher spielt hier die subjektive Wahrnehmung eines Zusammenhangs zwischen Funktionen eine Rolle.

Typische Repräsentanten solcher Beziehungen sind zum Beispiel das potenzielle Unterbrechen oder Abbrechen einer Funktion. Derartige Zusammenhänge zwischen Funktionen lassen sich besser anhand bestimmter Nutzungsmuster der Eingabehistorien, welche die Eingaben der untersuchten Funktionen tragen, festmachen. Eine differenzierte Betrachtung solcher Beziehungen erforderte im Detail die Untersuchung von Funktionszuständen beziehungsweise von Zustandsfolgen der Funktionen. In [Grü08] wird ein entsprechendes Zustandskonzept eingeführt, mit dem solche Zusammenhänge zwischen Funktionen formalisiert werden können. Wir werden deshalb in diesem Abschnitt nur die wesentlichen Aspekte erläutern und die Beziehungstypen vergleichsweise informell beschreiben.

Störung von Funktionen

Während im vorhergehenden Abschnitt der Einfluss auf den Zugang zu Funktionen im Zentrum des Interesses stand, gehen wir davon aus, dass die hier in Zusammenhang stehenden Funktionen ohne Einschränkung nutzbar sind. Wenn Funktionen durch andere Funktionen gestört werden können, hängt dies oft damit zusammen, dass sich die betroffenen Funktionen die zur Verfügung stehenden Ressourcen teilen. Aus Nutzungssicht sticht auch hier die Auswirkung von Funktionen mit höherer Priorität auf Funktionen mit niederer Priorität hervor, wie bereits bei der Diskussion des Beziehungstyps replace dargestellt wurde. Im Gegensatz zu den hier besprochenen Zusammenhängen, wird dort allerdings eine gemeinsam spezifizierte Eingabe vorausgesetzt.

Das Verhalten der abhängigen Funktion wird insofern beeinflusst, als dies durch einen Abbruch oder eine Unterbrechung der Interaktion sichtbar wird. Solange keine Funktion stört, kann die betrachtete Funktion stets ohne Einschränkung in der spezifizierten Art und Weise genutzt werden. Funktionen können aber von Funktionen mit höherer Priorität gestört werden, indem die Eingaben nicht mehr spezifikationsgemäß möglich sind. Die Steuerung des Kontrollflusses wird von der störenden Funktion übernommen und damit der betrachteten Funktion entzogen.

Abbruch einer Nutzungsfunktion

Der Abbruch einer Nutzungsfunktion ist ein typisches Nutzungsszenario, wobei das Eintreten eines Systemereignisses zur Nutzung einer anderen Funktion führt. Hierbei lässt sich der Effekt nicht unbedingt an einem im Vergleich zur Vorgabe abweichenden Verhalten nachweisen, sondern vielmehr anhand eines endlichen Nutzungsmusters beschreiben.

Definition 5.11 (Der Beziehungstyp cancel)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn die Nutzung der Funktion f den Abbruch einer in g definierten Interaktion verursacht, so schreiben wir

$$\text{`}f\text{'}\ \textbf{\textit{cancels}}\ \text{`}g\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{cancel}}(f, g)$$

□

Das Verhalten der betrachteten Funktion g entspricht stets der Spezifikation, solange die Nutzungskonventionen eingehalten werden. Ein Abbruch ist dann zu beobachten, wenn eine Eingabenachricht von g nicht mehr übertragen wird, obwohl diese gemäß Spezifikation zu erwarten wäre. Angenommen bis zu einem beliebigen Zeitintervall $n \in \mathbb{N}$ entspricht die Eingabe der Vorgabe in g. Das heißt, für ein $x \in \mathsf{dom}(s)$ gilt zunächst

$$(x{\downarrow_n})\,|\,I_g \in \mathsf{dom}(g)$$

Um von einer Störung sprechen zu können, nehmen wir $n > 0$ an. Der Spezifikation von g folgend erwartete man im nächsten Zeitintervall $(n+1)$ die Übertragung von Nachrichten gemäß einer Eingabehistorie der Domäne von g,

$$\exists\, x_g \in \mathsf{dom}(g) : x_g{\downarrow_n} = (x{\downarrow_n})\,|\,I_g \ \wedge\ \mathsf{ltm}(x_g) > n$$

was aber nicht der Fall ist.

$$(x{\downarrow_{n+1}})\,|\,I_g \notin \mathsf{dom}(g)$$

Wenn f nun die Ursache für das Abbrechen der in g definierten Interaktion ist, taucht in x ab dem Zeitintervall $(n+1)$ eine Eingabehistorie aus der Domäne von f auf.

$$(x{\uparrow_n})\,|\,I_f \in \mathsf{dom}(f)$$

Offensichtlich sind die hier angegebenen Formeln zwar notwendige, jedoch keine hinreichenden Bedingungen für eine cancel Beziehung zwischen f und g. Denn theoretisch kann auch eine andere Funktion als f für den Abbruch der Interaktion verantwortlich sein, bei

der ebenfalls ab dem Zeitintervall $(n + 1)$ die Übertragung ihrer Eingabenachrichten beobachtet werden kann. Insbesondere ist es so nicht möglich nachzuweisen, dass g tatsächlich nicht den erwünschten Folgezustand erreicht hat. Eine hinreichende Bewertung erforderte daher eine feinere, tiefer gehendere Untersuchung, bei der die Zustandsfolgen der Funktionen heranzuziehen sind.

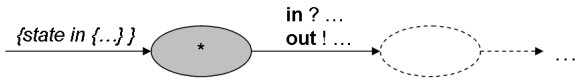


Abbildung 5.6: Schema einer cancel Beziehung im Zustandsübergangsdiagramm

In den Zustandsübergangsdiagrammen dieser Arbeit wird der Startzustand einer Nutzungsfunktion, welche potenziell eine andere stören kann, abkürzend mit * markiert. Damit soll zum Ausdruck gebracht werden, dass kein eindeutig identifizierbarer Kontrollzustand als Startzustand angegeben werden kann. Wie in der Abbildung 5.6 mit der Bedingung „*state in {...}*“ angedeutet, kann darüber hinaus allerdings die Menge der zulässigen Kontrollzustände eingeschränkt werden. Die Störung ist nur dann zulässig, wenn sich die abhängige Funktion in einem der angegebenen Kontrollzustände befindet. Die störende Funktion ist nur für die erfüllte Vorbedingung definiert.

Die Steuerung des Kontrollflusses wird von der störenden Funktion übernommen und wird der gestörten Funktion auch nicht mehr überlassen. Das heißt, jeder Endzustand der störenden Funktion ist ein beliebiger Kontrollzustand ungleich des Startzustands. Stehen zwei Funktion in einer cancel Beziehung, so kann die Nutzung einer Funktion tatsächlich von der anderen Funktion abgebrochen werden. Die gestörte Funktion wird nicht fortgesetzt.

Viele Mobiltelefone bieten eine Reihe von Anwendungen, die aus Nutzungssicht zunächst nicht mit den typischen Telefoniefunktionen in Verbindung zu bringen sind, wie zum Beispiel das Editieren von Nachrichten oder Adressen, Entertainment-Anwendungen oder auch das Navigieren in den Menüs zur Auswahl bestimmter Funktionen. Telefoniefunktionen benötigen nun dieselben Eingabekanäle (Tastatur) und Ausgabekanäle (Display, Lautsprecher) und besitzen in der Regel eine höhere Priorität als derartige Funktionen. Die Annahme eines eingehenden Anrufs resultiert beispielsweise in dem Abbruch der anderen Funktion, welche auch nach der Beendigung des Gesprächs nicht mehr fortgesetzt wird.

Unterbrechung einer Nutzungsfunktion

Ein eingehender Anruf muss allerdings nicht in jedem Fall zum Abbruch einer Funktion führen. Wird der Anruf etwa abgewiesen, so wird üblicherweise die gestörte Funktion an der Stelle fortgesetzt, an der sie unterbrochen wurde. Genau solche Zusammenhänge

können mit dem Beziehungstyp interrupt beschrieben werden. Auch die Unterbrechung einer Funktion ist ein spezielles Nutzungsszenario, welches wir ebenfalls anhand eines Nutzungsmusters veranschaulichen.

Definition 5.12 (Der Beziehungstyp interrupt)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn mit der Nutzung der Funktion f eine in g definierte Interaktion unterbrochen wird, so schreiben wir

$$\text{`}f\text{'}\ \textbf{\textit{interrupts}}\ \text{`}g\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{interrupt}}(f, g)$$

□

Wie beim Abbruchsszenario oben entspricht auch bei einer Unterbrechung bis zum Zeitintervall $n \in \mathbb{N}$ die Eingabe der Vorgabe. Ab dem Zeitintervall $(n+1)$ werden nicht die erwarteten Eingabenachrichten übertragen, wie sie in g spezifiziert sind.

$$\begin{array}{ll} & \exists\, x_g \in \mathsf{dom}(g) : x_g{\downarrow_n} = (x{\downarrow_n}) \,|\, I_g \\ \wedge & (x{\downarrow_{n+1}}) \,|\, I_g \notin \mathsf{dom}(g) \end{array}$$

Jedoch kann im Falle einer Unterbrechung ein Zeitintervall $k > n$ angeben werden, in dem die Nutzungsfunktion die unterbrochene Interaktion wieder fortsetzt.

$$(x{\uparrow_k}) \,|\, I_g = x_g{\uparrow_n}$$

Soweit zum Eingabemuster für eine betrachtete Funktion g. Beziehen wir nun eine unterbrechende Funktion sowie die Auswirkung auf die Ausgaben mit ein. Angenommen es gibt eine Funktion f, welche die Funktion g unterbricht. Das bedeutet, für eine Eingabe $x \in \mathsf{dom}(s)$ ist

$$s\dagger(I_f \cup I_g \blacktriangleright O_g).(x \,|\, I_f \cup I_g) \neq s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$$

Aber es existiert ein $x' \in \mathsf{dom}(g)$ mit

$$s\dagger(I_f \cup I_g \blacktriangleright O_g).(x \,|\, I_f \cup I_g) = s\dagger(I_g \blacktriangleright O_g).x'$$

wobei $(x \,|\, I_g)$ mithilfe des delay Operators auf x' abgebildet werden kann (siehe Definition 3.11). Wenn sich nun die Interaktion mit f über m Zeitintervalle erstrecke, ließe sich damit x' folgendermaßen bestimmen.

$$x' = \mathsf{delay}((x \,|\, I_g), \mathsf{ftm}(x \,|\, I_f), m)$$

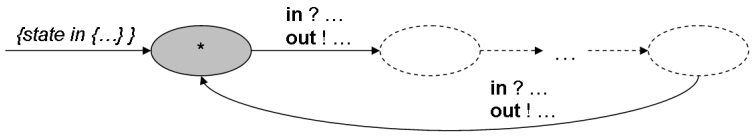


Abbildung 5.7: Schema einer interrupt Beziehung im Zustandsübergangsdiagramm

Mit der Fortsetzung der unterbrochenen Interaktion wird die Steuerung des Kontrollflusses wieder von der gestörten Funktion übernommen. Wie beim Beziehungstyp cancel kann auch bei unterbrechenden Funktionen kein eindeutig identifizierbarer Kontrollzustand als Startzustand angegeben werden. In Zustandsübergangsdiagrammen dieser Arbeit wird deshalb auch hier der Startzustand mit * markiert. Auch kann die abhängige Funktion nur dann gestört werden, wenn sie sich in einem der angegebenen Kontrollzustände befindet. Der wesentliche Unterschied ist in Abbildung 5.7 dargestellt, und zwar mit der Transition zurück in den mit * markierten Zustand. Der Startzustand und der Endzustand der störenden Funktion sind stets identisch.

Damit die Fortsetzung nach einer Unterbrechung gewährleistet werden kann, sollten bei unterbrechbaren Funktionen per Design geeignete Vorkehrungen zur Sicherung und Rekonstruktion lokaler Datenzustände getroffen werden. Dies kann etwa mit der Festlegung einer geeigneten internen Unterbrechungsschnittstelle, die zum Beispiel Methoden wie „pause“ und „resume“ beinhaltet, gelöst werden.

Auslösung anderer Nutzungsfunktionen

Eine Nutzungsfunktion kann nun nicht nur gestört werden, sondern auch nicht selten über die Nutzung einer anderen Funktion in Anspruch genommen werden. Der klassische Zusammenhang solcher Funktionen wird mit trigger umschrieben, um anzudeuten, dass die Nutzung einer Funktion ursächlich für die imminente Nutzung einer anderen Funktion ist.

Der Schwerpunkt liegt daher hier auf der Zugangsmöglichkeit zu der betrachteten Funktion g über eine Funktion f. Zwischen g und f besteht insofern eine Abhängigkeit, als f der Auslöser von g ist. Dies spielt insbesondere bei der Nutzung einer Funktion und der Bestimmung der involvierten Funktionen eine Rolle. Gerade bei der Verwendung von „getriggerten“ Funktionen können teilweise ungewollte Effekte entstehen [Sch04]. Im Rahmen eines Beispiels am Ende des Kapitels werden wir auf derartige Zusammenhänge noch einmal zurückkommen.

Wenn eine Funktion eine andere anstößt, so spielt es aus Nutzungssicht in der Tat keine Rolle, ob dies durch die direkte Kommunikation der Funktionen über interne Kanäle erfolgt. Genauso kann die Eingabe einer Nachricht eine geeignete Zustandsänderung des

umfassenden Softwaresystems bewirkt haben, das wiederum die Nutzung der angestoßenen Funktion zur Folge hat. Ist im Systementwurf eine direkte Kommunikation vorgesehen, legt dies bereits einen architekturellen Zusammenhang der Funktionen fest, da eine geeignete Interaktionsschnittstelle zu definieren ist. Die Übertragung dieser internen Nachrichten ist aus Nutzersicht allerdings nicht zu beobachten.

Auch aus der Perspektive der übertragenen Nachrichten auf den Schnittstellen ist eine trigger Beziehung zwischen zwei Funktionen eher unspektakulär. Insbesondere an den Schnittstellen der Funktionen lässt sich kein von den Vorgaben abweichendes Verhalten erkennen. Wie die anderen Beziehungstypen dieses Abschnitts korrespondiert auch das Anstoßen einer Funktion mit einem bestimmten Nutzungsszenario.

Definition 5.13 (Der Beziehungstyp trigger)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn die Funktion f die Nutzung der Funktion g initiiert und das Verhalten von f nicht vom Resultat der Interaktion mit g abhängt, so schreiben wir

$$\text{`}f\text{'}\ \textbf{\textit{triggers}}\ \text{`}g\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{trigger}}(f, g)$$

□

Stehen Funktionen in einer trigger Beziehung, so soll damit zum Ausdruck gebracht werden, dass es Eingabehistorien einer Funktion gibt, die als Konsequenz stets Eingaben einer anderen Funktion nach sich ziehen. Bei 'f' **triggers** 'g' folgen den Nachrichten einer bestimmten Eingabe von f stets Eingabenachrichten von g.

Die Menge der Eingabehistorien in einem Supersystem s, bei denen die Eingabenachrichten von g stets nach den Eingabenachrichten von f übertragen werden, sei mit P_{SEQ} bezeichnet.

$$P^s_{SEQ}(f, g) = \{x \in P^s_V(f) \cap P^s_V(g) \ : \mathsf{ltm}(x \,|\, I_f) < \mathsf{ftm}(x \,|\, I_g)\}$$

Steht nun f in einer trigger Beziehung zu g, so ist $P^s_{SEQ}(f, g)$ notwendigerweise nicht leer.

$$\text{`}f\text{'}\ \textbf{triggers}\ \text{`}g\text{'}\ \textbf{in}\ s \ \Rightarrow\ P^s_{SEQ}(f, g) \neq \varnothing$$

Dies ist gewiss ein schwaches Kriterium, da sich der kausale Zusammenhang zwischen den Eingaben der Funktionen kaum erkennen lässt. Gemäß des angegebenen Prädikats müssen nämlich die Eingaben für g nur in einem beliebigen Zeitintervall nach der letzten Eingabenachricht für f übertragen werden.

Für eine eindeutigen Nachweis ist auch hier im Grunde wieder eine detaillierte Betrachtung der Zustandsfolgen der Funktionen erforderlich. Eine stärkere Formulierung erreichte man aber auch, indem beispielsweise eine geeignete Relation $R^s_C(f, g)$ eingeführt wird,

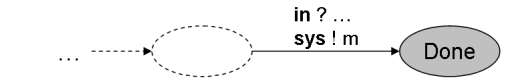
welche die tatsächlich kausal abhängigen Eingabehistorien der beiden Funktionen enthält.

$$R^s_C(f,g) = \left\{ \begin{array}{rl} (x_f, x_g) \in (\mathsf{dom}(f), \mathsf{dom}(g)) & : \\ \exists\, x \in \mathsf{dom}(s) & : \quad x_f = (x \,|\, I_f) \;\Rightarrow\; x_g = (x{\uparrow_{\mathsf{ltm}(x_f)}} \,|\, I_g) \end{array} \right\}$$

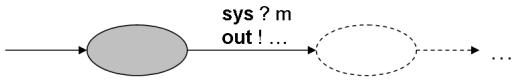
Eine Eingabe x_g ist in s kausal abhängig von einer Eingabe x_f. In der oben angegebenen Definition von $P^s_{SEQ}(f,g)$ ist dann das Prädikat $\mathsf{ltm}(x \,|\, I_f) < \mathsf{ftm}(x \,|\, I_g)$ durch

$$(x \,|\, I_f, x \,|\, I_g) \in R^s_C(f,g)$$

zu ersetzen.



(a) Triggernde Funktion



(b) Getriggerte Funktion

Abbildung 5.8: Schema einer trigger Beziehung im Zustandsübergangsdiagramm

In Abbildung 5.8 ist eine Art der Modellierung von trigger Beziehungen schematisch dargestellt, wie wir sie auch in dieser Arbeit verwenden werden. Die nutzende Funktion in Abbildung 5.8(a) gibt eine Nachricht m auf einem internen Kanal **sys** aus, bevor ein Endzustand, der hier mit *Done* markiert ist, erreicht wird. Die angestoßene Funktion in Abbildung 5.8(b) wiederum erhält im Startzustand ebenfalls auf dem Kanal **sys** diese Nachricht m als Eingabe. m ist gleichsam eine auslösende Nachricht für weitere Interaktionen der genutzten Funktion.

Obgleich eine trigger Beziehung vergleichsweise unkritisch erscheint, kann diese allerdings im Zusammenhang mit anderen Funktionsbeziehungen zu einem inkonsistenten Design führen. Angenommen es gibt drei Funktionen f, g und h, wobei g die Voraussetzung für den Zugang zur Funktion h schaffe ('g' **prepares** 'h'). Wenn f nun g triggern sollte, kann auch f mit h nicht ohne geeignete Maßnahmen kombiniert werden, obwohl sie im Grunde möglicherweise ohne Einschränkung kombinierbar wären.

Indirekte Steuerung des Kontrollflusses

Abschließend betrachten wir erneut die Situation, dass die Interaktion einer Nutzungsfunktion von einer anderen Nutzungsfunktion abhängt (siehe Beziehungstyp direct). Hier allerdings ist die betrachtete Funktion selbst ursächlich dafür verantwortlich, dass ihr

Verhalten unter anderem von einer anderen Funktion abhängt. Tatsächlich soll bei dieser Art des Zusammenhangs zum Ausdruck gebracht werden, dass es Eingaben der betrachteten Funktion g gibt, die zu bestimmten Interaktionsmustern mit einer Funktion f führen, welche wiederum das Verhalten von g beeinflussen. Wir verwenden in diesem Fall zur Bezeichnung des Beziehungstyps consult (im Sinne von „heranziehen“ oder „hinzuziehen“).

Wie die Funktionen letztlich interagieren, wird mit dem consult Beziehungstyp nicht festgelegt. Aus der Blackboxsicht kann dies durchaus auch über Nachrichten auf internen Kanälen erfolgen. Entscheidend ist jedoch, dass für beide Funktionen eine externe Schnittstelle existiert. Sonst kann auch kein entsprechendes Verhalten speziell der konsultierten Funktion f beobachtet werden und kein Zusammenhang von g zu f hergestellt werden. Aus Nutzungssicht ist f dann auch keine relevante Nutzungsfunktion.

Definition 5.14 (Der Beziehungstyp consult)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wenn die Funktion g die Nutzung der Funktion f initiiert und das Verhalten von g abhängig vom Resultat der Interaktion mit f ist, so schreiben wir

$$\text{`}g\text{'}\ \textbf{\textit{consults}}\ \text{`}f\text{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{consult}}(g, f)$$

□

Aus der Sicht eines Nutzers wird die Interaktion einer konsultierten Funktion f nicht getrennt, sondern eher als zur betrachteten Funktion g gehörige Ein- und Ausgabe wahrgenommen. Insbesondere erscheint das Verhalten von g deterministisch, da jede den Konventionen entsprechende Eingabehistorie mit den Eingaben von g und f eine eindeutige Ausgabehistorie nach sich zieht. Theoretisch kann man nun ausschließlich g betrachten, indem man die Eingaben und Ausgaben von f ausblendet und die Historien auf $(I_g \blacktriangleright O_g)$ projiziert. Die Ausgaben von g können dadurch nicht mehr eindeutig sein. Beziehungsweise umgekehrt formuliert: die Menge der Ausgabehistorien auf O_g reduziert sich, sobald Eingaben auf I_f mit betrachtet werden. Für eine Eingabehistorie $x \in \mathsf{dom}(s)$ gilt:

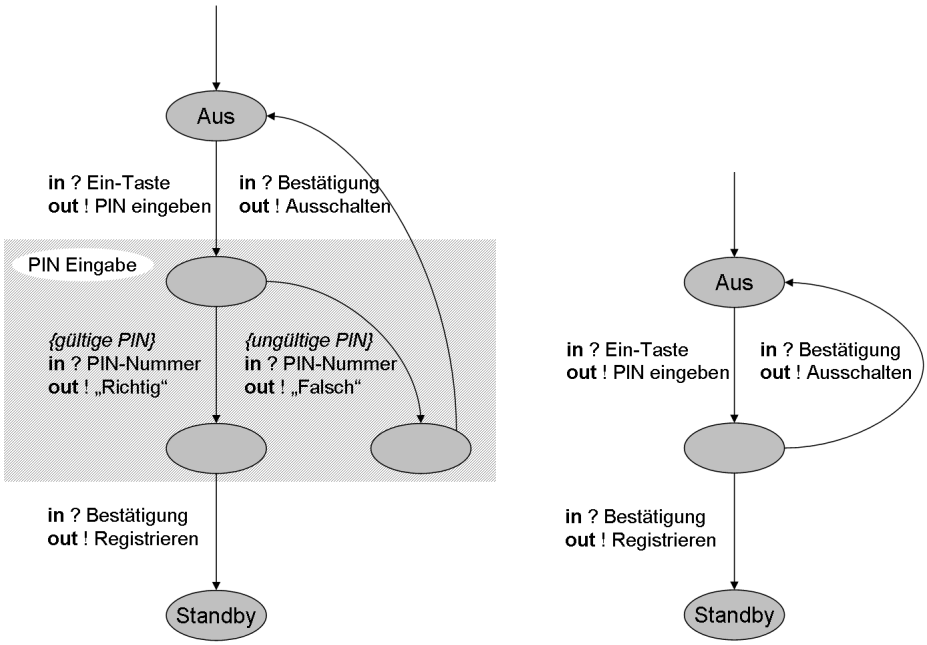
$$s\dagger(I_f \cup I_g \blacktriangleright O_g).(x \,|\, I_f \cup I_g) \subset s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$$

(siehe auch Beziehungstyp refine). Darüber hinaus gilt aber noch folgender Zusammenhang zwischen den Ausgabehistorien der Funktionen. Für alle $y \in s\dagger(I_g \blacktriangleright O_g).(x \mid I_g)$ gibt es ein $y' \in \mathsf{ran}(f)$ mit

$$y' \in (s.x) \,|\, O_f \ \Rightarrow\ y \in (s.x) \,|\, O_g$$

Die Ausgaben von f bestimmen eindeutig die Ausgaben auf O_g. Sie selektieren gleichsam die Ausgaben aus der Menge $s\dagger(I_g \blacktriangleright O_g).(x \,|\, I_g)$ der möglichen Ausgaben.

Betrachten wir dazu als Beispiel eine einfache Spezifikation für das Einschalten eines Mobiltelefons.



(a) Aus der Nutzerperspektive

(b) Abstraktion von der konsultierten Funktion «PIN Eingabe»

Abbildung 5.9: Einschalten eines Mobiltelefons mit «PIN Eingabe»

Das Einschalten eines Mobiltelefons nur dann erfolgreich, wenn eine korrekte PIN eingegeben wird. «Einschalten» konsultiert die Funktion «PIN Eingabe», wobei wir annehmen, dass für «Einschalten» als Reaktion auf «PIN Eingabe» mindestens zwei verschiedene Ausgaben zu beobachten sind, je nachdem ob die richtige oder eine falsche PIN Nummer eingegeben wurde. Aus der Perspektive eines Nutzers ist das Verhalten etwa gemäß des Zustandsübergangsdiagramms in Abbildung 5.9(a) spezifiziert.

Abstrahierte man nun von den Interaktionen der Funktion «PIN Eingabe», die in Abbildung 5.9(a) grau unterlegt ist, entspricht dies der Spezifikation in Abbildung 5.9(b). Man erkennt dort leicht die Mehrdeutigkeit der Funktion.

Einerseits wird durch das Konsultieren einer Funktion die Interaktion der betrachteten Nutzerfunktion gesteuert. Andererseits wird dadurch gewissermaßen auch die Unterbestimmtheit einer Funktion eliminiert. Insofern lässt sich eine Ähnlichkeit zum refine Beziehungstyp erkennen. Wie bei direct Beziehungen wird der Kontrollfluss gesteuert, hier allerdings abhängig vom Ergebnis der konsultierten Funktion. Bei einer consult Beziehung

ist die betrachtete Funktion selbst für die Beeinflussung durch die konsultierte Funktion verantwortlich. Dies ist selbstverständlich ein gewünschter Effekt, wenn ein architektureller Zusammenhang besteht.

Obwohl es bei consult und trigger Beziehungen nahe liegt, müssen nicht notwendigerweise stets entsprechende Interaktionsschnittstellen zwischen solchen Funktionen definiert werden. Wenn dennoch das Funktionsverhalten auf einem architekturellen Zusammenhang beruht, entspricht dies einer Funktionskomposition. Im Unterschied zu einer beliebigen Kombination von Funktionen sind Komponenten gewissermaßen „vor-integriert“. Das drückt sich zum Beispiel dadurch aus, dass bereits entsprechende Zustandsübergänge und Ausgaben der nutzenden Funktion definiert sind, die etwa als Fehlermeldungen zu beobachten sind, wenn die konsultierte Funktion nicht wie vorgesehen reagiert. Dies kann dann auch die Reaktion auf die potenzielle Nichtverfügbarkeit einer zu triggernden oder zu konsultierenden Funktion beinhalten.

5.4 Zusätzliche Verhaltensmuster in Kombination

Die Erweiterung von bestehender Funktionalität mit optional zur Verfügung stehenden Funktionen ist ein gängiges Konzept, wie es zum Beispiel bereits von der UML zur Modellierung von Nutzungsfällen angeboten wird. Im Hinblick auf Funktionsbeziehungen aus Nutzungssicht sind allerdings die konkreten Erweiterungsbedingungen von untergeordnetem Interesse. Letzten Endes ist eine derartige Erweiterung auf die Ergänzung eines Interaktionsmusters zurückzuführen. Neben den vorgegebenen Interaktionsmustern der betrachteten Funktion sind zusätzliche Interaktionsmuster zulässig, die sich aus den spezifizierten Interaktionsmustern beider Funktionen zusammensetzen.

Eine weitere Ausprägung werden wir unter dem Begriff der Zugangserweiterung kennen lernen. Darunter verstehen wir die Schaffung zusätzlicher Eingabemöglichkeiten in der Kombination von Funktionen zu bereits definierten Ausgaben einer betrachteten Funktion. Dies wurde bereits im Zusammenhang mit der Definition der Menge undefinierter Eingaben $P_0^s(g)$ für eine betrachtete Funktion g (siehe Seite 76) erwähnt. In $P_0^s(g)$ gibt es möglicherweise Eingabehistorien, bei denen zwar Nachrichten auf Kanälen von I_g übertragen werden, die jedoch in g nicht definiert sind.

Definition 5.15 (Der Beziehungstyp extend)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wir schreiben

$$\textit{`}f\textit{'}\ \textbf{\textit{extends}}\ \textit{`}g\textit{'}\ \textbf{\textit{in}}\ s \quad \textit{oder} \quad \textbf{\textit{extend}}(f, g)$$

wenn f und g keine gemeinsamen Eingaben besitzen und es eine Eingabe für f gibt, die zu einer in g definierten Ausgabe führt. □

Im dem Fall gibt es eine Eingabehistorie $x \in \mathsf{dom}(s)$, die keine für g definierte Eingabe und zugleich eine für f definierte Eingabe enthält. Ist dann eine Ausgabe aus dem Wertebereich von g an O_g zu beobachten, sprechen wir auch von einer Zugangserweiterung über f. Es gibt eine Eingabehistorie $x \in P_0^s(g)$, für die gilt

$$x \in P_V^s(f) \ \Rightarrow\ s\dagger(I_g \cup I_f \blacktriangleright O_g).(x \,|\, I_g \cup I_f) \subseteq \mathsf{ran}(g)$$

Aufgrund der subjektiven Wahrnehmung des Zusammenhangs bewerten wir ferner g als nicht unabhängig von f, wenn f eine Zugangserweiterung für g bietet (vergleiche auch die Kriterien für die Unabhängigkeit auf den Seiten 96 bis 98).

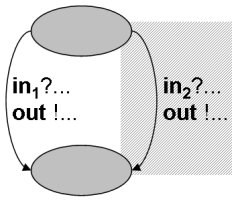


Abbildung 5.10: Schema einer extend Beziehung im Zustandsübergangsdiagramm

In dem Zustandsübergangsdiagramm in Abbildung 5.10 ist eine Zugangserweiterung in der Kombination von Funktionen dargestellt. Der Teil, für den die beeinflussende Funktion verantwortlich ist, ist rechteckförmig unterlegt. Zusätzlich mögliche Eingaben führen zu denselben Ausgaben und damit zur gleichen Reaktion des Softwaresystems. In der Abbildung ist dies an der Verwendung derselben Ausgabekanalbezeichnung **out** bei beiden Transitionen zu erkennen.

Diese Art des Zusammenhangs lässt sich am Beispiel von Mobiltelefonfunktionen verdeutlichen. Ein Anruf wird üblicherweise durch Drücken der Abheben-Taste angenommen. Die Funktion «Anruf annehmen» kann nun durch die Funktion «Jede Taste» so erweitert werden, dass die Annahme im Grunde mit einer beliebigen Taste möglich ist. Die Ausgaben ändern sich dabei jedoch nicht.

Ergänzung von Interaktionsmustern

Durch das Kombinieren mit anderen Funktionen können bereits definierte Interaktionsmuster einer betrachteten Funktion zu komplexeren Interaktionsmustern ergänzt werden. Diese Eigenschaft ist jedoch eine grundlegend andere im Vergleich zum extend Beziehungstyp.

Definition 5.16 (Der Beziehungstyp complement)

Zwei kombinierbare Funktionen f und g seien eingebettet in ein Supersystem s. Wir schreiben

$$\text{`}f\text{'} \textbf{ complements } \text{`}g\text{'} \textbf{ in } s \quad \textit{oder} \quad \textbf{complement}(f, g)$$

wenn eine in g definierte Interaktion an einer eindeutig festgelegten Stelle durch eine in f definierte Interaktion ergänzt wird. □

Diese Ausprägung einer Funktionsbeziehung ist eher im Zusammenhang mit einem Nutzungsszenario zu interpretieren. Insofern ist es anschaulicher, den Effekt weniger im Hinblick auf das Verhalten der Funktionen zu beschreiben, sondern anhand eines entsprechenden Nutzungsmusters darzustellen.

Hier wird gleichsam eine in g definierte Eingabesequenz erweitert. Es gibt eine Eingabehistorie $x \in \mathsf{dom}(s)$, welche sowohl eine für g, als auch eine für f definierte Eingabe enthält,

$$x \in P_V^s(g) \cap P_V^s(f)$$

sowie ein festgelegtes Zeitintervall $n \in \mathbb{N}$, in dem die Übertragung der Eingaben von f beginnt.

$$\exists\, x_g \in \mathsf{dom}(g), x_f \in \mathsf{dom}(f) : (x{\downarrow_n})\,|\,I_g = x_g{\downarrow_n} \;\wedge\; (x{\uparrow_{n+ltm(x_f)}})\,|\,I_g = x_g{\uparrow_n}$$

Zwischen den Zeitintervallen n und $n+ltm(x_f)$ werden Nachrichten gemäß der Vorgabe in f übertragen. In den Zeitintervallen davor und danach entspricht die Eingabe der Vorgabe in g. Offenkundig ist dies eine sehr spezielle Festlegung. Sie soll aber die wesentliche Konstruktion einer derartigen Beziehung verdeutlichen.

Die verwendete Schreibweise erinnert augenscheinlich an die Diskussion des interrupt Beziehungstyps. Im Unterschied dazu ist die Stelle n allerdings eindeutig festgelegt. Außerdem ist $n = 0$ oder $n = ltm(x_g)$ ausdrücklich zulässig, sprich: eine Ergänzung nach beziehungsweise vor einer gegebenen Eingabe von g.

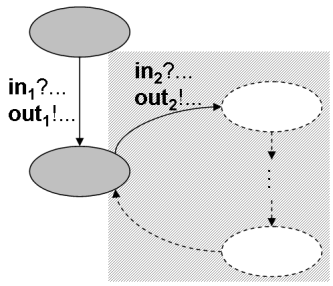


Abbildung 5.11: Schema einer complement Beziehung im Zustandsübergangsdiagramm

Das Zustandsübergangsdiagramm in Abbildung 5.11 zeigt ein Schema für die Erweiterung eines bestehenden Interaktionsmusters. Zusätzliche Transitionen aus bestimmten Kontrollzuständen heraus führen gleichsam zu einer Ergänzung desselben. Rechteckförmig unterlegt ist hierbei wieder der Teil, für den die beeinflussende Funktion verantwortlich ist.

Das Navigieren in Menüs ist eine typische Funktion, zu der eine Reihe anderer Funktionen in einer complement Beziehung stehen. Beispielsweise kann ein bestehendes Menü für die Auswahl weiterer Funktionen durch geeignete Einträge ergänzt werden. Die Nutzung der Funktionen wird mit entsprechenden Erweiterungen der Eingabe- und Ausgabehistorien festgelegt.

5.5 Zusammenfassung

Mit den in diesem Kapitel und in Kapitel 4 vorgestellten Beziehungstypen lassen sich Funktionszusammenhänge eines Systems hinreichend beschreiben. Zweifelsohne ließe sich eine Vielzahl weiterer Bezeichnungen für Funktionsbeziehungen finden. Diese sind aber entweder nicht im Fokus dieser Arbeit, etwa entwicklungsmethodische Beziehungen wie „Äquivalenz“ oder „Gleichheit“ von Funktionen, die Bezeichnungen werden mehr oder weniger synonym verwendet, oder die Bedeutungen der Beziehungstypen sind letztendlich auf die genannten Beziehungstypen zurückzuführen.

Konsequenzen von Funktionsbeziehungen

Alleine mit der Festlegung einer Beziehung zwischen Funktionen ist noch nichts über deren technische Umsetzung gesagt. Eine Beziehung abstrahiert gewissermaßen das beobachtbare, möglicherweise beeinflusste, Verhalten einer Funktion in Kombination mit einer anderen eingebetteten Funktion. Anhand von Zustandsautomaten als Mittel zur Spezifikation von Funktionen wurden zwar mögliche Konsequenzen, zum Beispiel mit

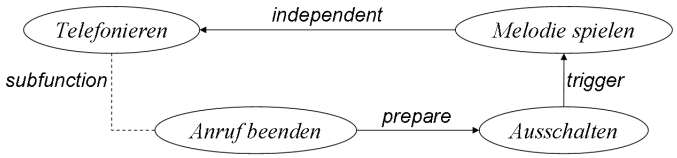
der Einführung von Vorbedingungen und Nachbedingungen, aufgezeigt. Damit ist aber zugleich in keinster Weise festgelegt, wie der Informationsaustausch unter den betroffenen Funktionen umzusetzen ist. Dies kann selbstverständlich auf direktem Wege über eine geeignet definierte interne Schnittstelle erfolgen. Aber auch die Nutzung gemeinsamer dritter Funktionen ist denkbar, genauso wie die Integration in Software „Frameworks“, über die sämtlicher Austausch von Information läuft.

Funktionen müssen nicht immer exklusiv in genau einer Art von Beziehung stehen. Wenn der Zusammenhang für ein Funktionenpaar durch mehrere verschiedenartige Beziehungen definiert wird, ist dabei zu beachten, dass dann die Beziehungstypen nicht im Widerspruch stehen. Dies gilt insbesondere auch für indirekte Beziehungen zwischen Funktionen, die sich erst aus gegebenen Beziehungen ableiten lassen. Bei einer bestimmten Serie eines Mobiltelefons führte das unkontrollierte Abspielen einer Melodie bereits kurz nach dessen Einführung zu einem Verkaufsstopp. In [Sch04] wurde dieses Verhalten zwar allgemein als ein Softwarefehler eingeordnet. Die Ursache kann aber ebenso ein nicht erkannter Zusammenhang zwischen den Nutzungsfunktionen sein. Zur Illustration betrachten wir im folgenden Beispiel einen vereinfachten Ausschnitt einer Nutzungsarchitektur.

Beispiel (Indirekte Beziehungen zwischen Funktionen eines Mobiltelefons)

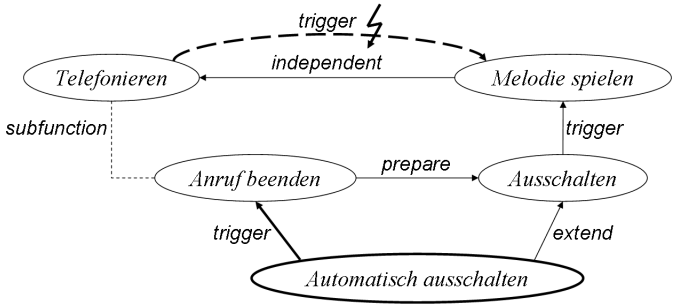
Wir betrachten zunächst die Zusammenhänge der Funktionen «Telefonieren», «Anruf beenden», «Melodie spielen» und «Ausschalten». Die Funktion «Anruf beenden» ist eine Subfunktion von «Telefonieren» und beim Ausschalten wird automatisch das Abspielen einer Ausschaltmelodie initiiert, das mit der Beziehung ‘*Ausschalten*’ **triggers** ‘*Melodie spielen*’ beschrieben wird.

Das Gerät soll darüber hinaus nicht während eines Telefonats ausgeschaltet werden können. «Anruf beenden» und «Ausschalten» stehen deshalb nur in verschiedenen Betriebsmodi zur Verfügung, das mit ‘*Anruf beenden*’ **prepares** ‘*Ausschalten*’ ausgedrückt wird. Schließlich darf während eines Telefonats unter keinen Umständen eine Melodie abgespielt werden, da sich der Lautsprecher am Ohr befindet und lautes Abspielen zu Hörschäden führen kann. Diese Anforderung wird mit **independent**(*Melodie spielen*, *Telefonieren*) dokumentiert. Zusammengefasst ergibt sich also die folgende Nutzungsarchitektur:



Nun gibt es nicht nur die Möglichkeit, dass das Gerät manuell per Tastatureingabe durch den Anwender ausgeschaltet wird, sondern auch automatisch zu einer bestimmten Uhrzeit oder bei unzureichender Energieversorgung. Diese Funktion «Automatisch ausschal-

ten» erweitert das Ausschalten um die entsprechenden Zugangsmöglichkeiten. Darüber hinaus benutzt sie die Funktion «Anruf beenden», da das Ausschalten des Geräts während eines Gesprächs nicht möglich ist, '*Automatisch ausschalten*' **triggers** '*Anruf beenden*'. Mit der neuen Funktion ergibt sich folgendes Bild:



Dadurch, dass «Automatisch ausschalten» über das Anstoßen von «Anruf beenden» selbst die Voraussetzung für das Ausschalten schafft und beim Ausschalten das Spielen der Ausschaltmelodie angestoßen wird, triggert in diesem Szenario die Funktion «Anruf beenden» gleichsam das Abspielen. Deshalb kann man sich als abgeleitete Beziehung zwischen «Telefonieren» und «Melodie spielen» etwa den Beziehungstyp trigger vorstellen. Diese Beziehung stünde nun aber im Widerspruch zu der angenommenen Unabhängigkeit. □

Eine automatisierte Verifikation einer Nutzungsarchitektur im Sinne der Identifizierung widersprüchlicher Beziehungen, seien sie direkt definiert oder auch abgeleitet, liegt zwar nicht im Fokus dieser Arbeit, sollte aber für ein hinreichend methodisches Vorgehen unbedingt angestrebt werden.

Organisatorische Beziehungen

An dieser Stelle sei noch kurz das Thema der organisatorischen Beziehungen aufgegriffen. Mit einem Funktionen-Bundle kann ausgedrückt werden, dass zwei Funktionen unbedingt gemeinsam zu betrachten sind, und zwar losgelöst von der hierarchischen Strukturierung der Funktionen mittels der Subfunktionsrelation. Die Kombinierbarkeit der Funktionen ist damit obligatorisch. Betrachtet man lediglich das Verhalten der Funktionen, kann dieser Fall selbstverständlich gegeben sein, wenn eine Funktion eine andere Funktion gewissermaßen voraussetzt. Eine bundle Beziehung entspricht dann zum Beispiel einer notwendigen (aber nicht hinreichenden) Bedingung für eine enable Beziehung, zum Beispiel:

$$\mathbf{enable}(f, g) \vee \mathbf{enable}(g, f) \Rightarrow \mathbf{bundle}(f, g).$$

Mit bundle Beziehungen sind aber auch statische, verhaltensunabhängige Abhängigkeiten

denkbar, wie sie beispielsweise bei Featurebäumen mit AND-Kanten angegeben oder bei Produktdefinitionen einer Produktlinie vorgegeben sein können. In diesem Sinne ist diese Art von Beziehung eher als Vorgabe zu sehen, deren Einhaltung durch entsprechende Bedingungen zu überprüfen ist. Wenn es beispielsweise die nichtfunktionale Anforderung gibt, dass zwei Funktionen nicht getrennt voneinander zur Verfügung gestellt werden dürfen, ist dies widersprüchlich zu deren Nicht-Kombinierbarkeit.

Die Beziehungstypen und ihre Zusammenhänge

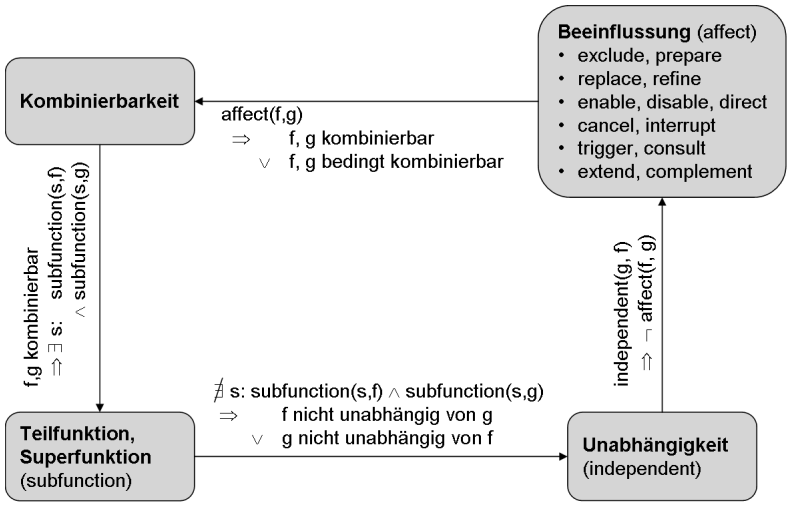


Abbildung 5.12: Beziehungstypen im Überblick

Bevor wir uns nun im Anschluss im Rahmen einer Fallstudie im Detail mit den Beziehungstypen auseinandersetzen, fassen wir noch einmal zusammen. In der Übersicht in Abbildung 5.12 sind alle in dieser Arbeit identifizierten Beziehungstypen sowie deren Zusammenhänge untereinander erfasst.

Für die in dieser Arbeit definierten Beziehungstypen, welche den Einfluss einer Funktion auf eine andere beschreiben, wird die Kombinierbarkeit oder bedingte Kombinierbarkeit der Funktionen notwendigerweise vorausgesetzt. Dies ist durch die Verbindung im oberen Teil der Grafik angezeigt. Die Umkehrung der Implikation gilt allerdings nicht, denn es gibt kombinierbare Funktionen, die sich nicht beeinflussen und zum Beispiel unabhängig voneinander sind.

Der Pfeil auf der linken Seite der Grafik spiegelt die Definition kombinierbarer Funktionen wieder (siehe Definition 4.9). Die Kombinierbarkeit zweier Funktionen ist äquivalent dazu, dass von einem Softwaresystem der Zugang zu den Funktionen angeboten wird. Die gilt selbstverständlich auch für den Fall, dass die Funktionen in einer Teilfunktions-

relation stehen, da insbesondere trivial kombinierbare Funktionen ebenfalls kombinierbar sind.

Der dargestellte Zusammenhang auf der rechten Seite ist äquivalent zu der Implikation in Definition 5.1. Wenn eine Funktion unabhängig in ihrem Verhalten von einer anderen Funktion ist, so nimmt diese folglich auch keinen Einfluss auf deren Verhalten.

Schließlich fällt noch der Zusammenhang zwischen Subfunktionen eines Systems und der Unabhängigkeit dieser Subfunktion im unteren Teil der Grafik auf. Wenn es kein System gibt, das den Zugang zu zwei Funktionen zugleich anbieten kann, dann muss es wenigstens eine Abhängigkeit zwischen diesen Funktionen geben, die zu widersprüchlichen Ausgabehistorien führt.

Aus Gründen der Übersichtlichkeit sind in der Grafik nicht alle Verbindungen eingezeichnet. Dennoch sei erwähnt, dass wenn zwei Funktionen in ihrem Verhalten wechselseitig unabhängig voneinander sind, daraus trivialerweise deren Kombinierbarkeit folgt. Denn die Ausgaben jeder Funktion sind stets vorgabegemäß, selbst wenn sich die Eingaben für die Funktionen beliebig überlappen. In der Kombination solcher Funktionen können daher auch keine widersprüchlichen Ausgaben entstehen.

Wird dagegen eine Funktion durch eine andere Funktion beeinflusst, können sich die Funktionen unter Umständen gegenseitig ausschließen, wenn es nicht miteinander vereinbare Ausgaben gibt. Dennoch können die Funktionen für bestimmte Eingaben kombinierbar sein. Selbst wenn es Eingaben der Superfunktion gibt, die widersprüchliche Ausgaben zur Folge haben, kann die Domäne auf eine Teilmenge von Eingabehistorien geeignet eingeschränkt werden. Unter der Bedingung der Existenz einer solchen Teilmenge von zulässigen Eingabehistorien sind die Funktionen (bedingt) kombinierbar.

Kapitel 6

Fallstudie

In diesem Abschnitt werden die Zusammenhänge von Funktionen anhand eines größeren konkreten Beispiels untersucht. Dies dient einerseits als Basis für die Erläuterung verschiedener Abhängigkeiten, um sie mithilfe konkreter Spezifikationen zu verdeutlichen. Andererseits werden damit die vorgestellten Konzepte praktisch erprobt.

Für die Analyse wurde das System *Mobiltelefon* ausgewählt. Viele Anwender sind mit den typischen Funktionen eines Mobiltelefons vertraut oder kennen zumindest das Verhalten aus eigener Erfahrung. Daher können wir uns direkt auf die wesentlichen Aspekte konzentrieren, ohne erst eingehend die Funktionalitäten erörtern zu müssen. Daneben ist die Funktionalität eines Mobiltelefons überschaubar, aber zugleich umfangreich genug, um alle Konzepte spiegeln zu können. Nachdem sich der Umfang der Funktionalität eines Standardmobiltelefons in den letzten Jahren stetig vergrößert hat (zum Beispiel SMS, MMS, CB, WAP, Diktiergerät, Spiele, Kamera und so weiter) würde eine Gesamtbetrachtung den Rahmen dieser Arbeit sprengen. Mit der erweiterten Funktionalität sind jedoch keine Zusammenhänge zwischen Funktionen verbunden, welche die Definition weiterer Beziehungstypen erfordert. Deshalb wurde die Anzahl begrenzt und eine Reihe entsprechend repräsentativer Funktionen ausgewählt.

Wir beschreiben zunächst in kompakter Form den gewählten Funktionsumfang. Im Anschluss wird der Übergang von isolierten zu kombinierten Funktionsspezifikationen gezeigt, bei dem die zugrunde liegenden geforderten Beziehungen zwischen den Funktionen ausschlaggebend sind. Abschließend werden die Zusammenhänge der Funktionen im Rahmen einer Nutzungsarchitektur dargestellt.

6.1 Die Funktionen im Überblick

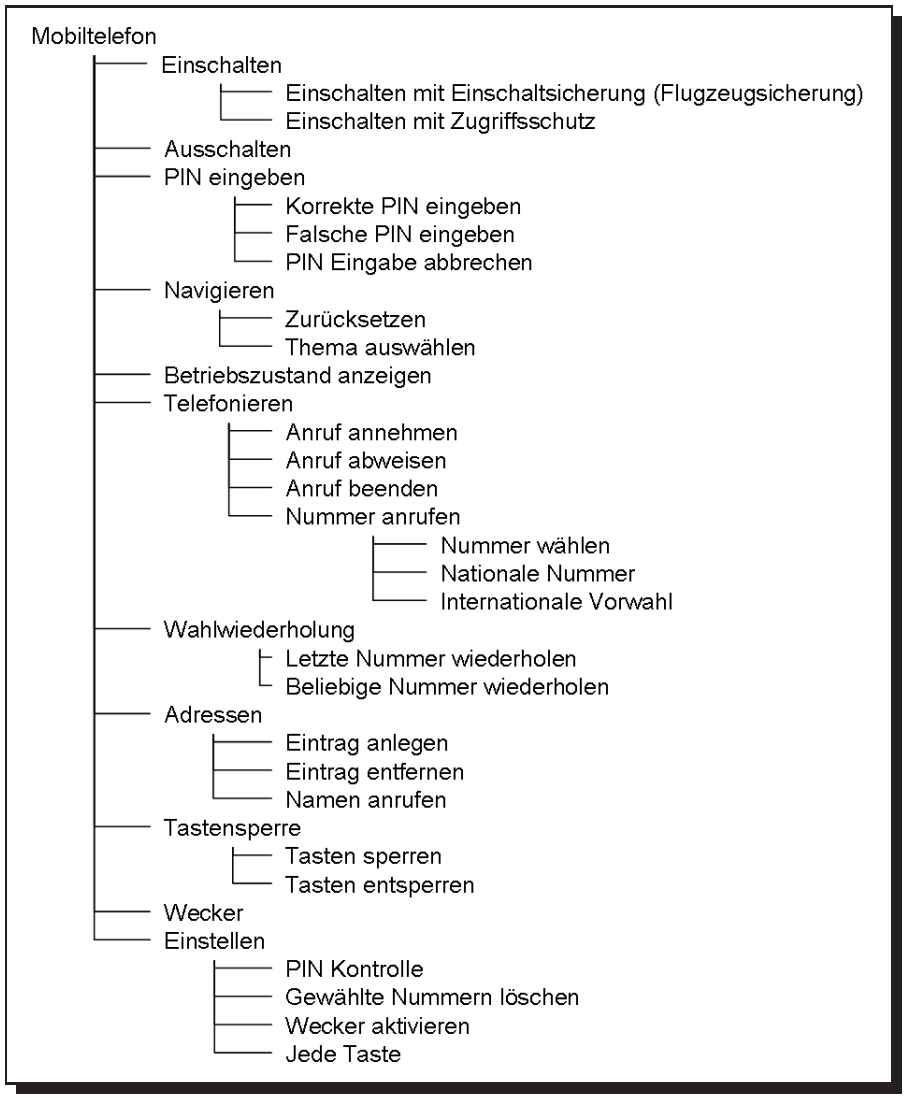


Abbildung 6.1: Ausgewählte Nutzungsfunktionen eines Mobiltelefons

In Abbildung 6.1 sind die für diese Arbeit ausgewählten Nutzungsfunktionen eines Mobiltelefons dargestellt[1]. Die hierarchischen Zusammenhänge der Funktionen sind dort

[1] Die Funktionen lassen sich auch ohne reales Gerät komfortabel nachvollziehen, zum Beispiel mithilfe des „Mobile Toolkit“ von BenQ/Siemens [MTK06]. Einige der folgenden Darstellungen wurden ebenfalls von diesem Toolkit übernommen.

bereits zu erkennen. Die Verbindungslinien in der Übersicht stellen Teilfunktionsbeziehungen dar, beispielsweise gilt

subfunction(*Mobiltelefon*, *Telefonieren*)
und **subfunction**(*Telefonieren*, *Anruf annehmen*)

und so weiter.

Informelle Beschreibung der Mobiltelefonfunktionen

Vor der weiteren Diskussion der Beziehungen zwischen den Funktionen werden diese nun im Einzelnen beschrieben, wobei jeweils die Erklärung absichtlich kurz gehalten ist. Wir haben bewusst darauf verzichtet, weniger oft benutzte Funktionen mit einzubeziehen. Die Beschreibung der einzelnen Funktionalitäten dient an dieser Stelle deshalb lediglich dem gemeinsamen Verständnis.

«Einschalten»
: Durch langes Drücken der Auflegen-Taste wird das Gerät eingeschaltet und in den betriebsbereiten Zustand versetzt. Ist der Zugriffsschutz deaktiviert, erfolgt ersatzweise eine Sicherheitsabfrage, um ein versehentliches Einschalten zu verhindern.

«Ausschalten»
: Wenn das Gerät betriebsbereit ist, wird durch langes Drücken der Auflegen-Taste das Gerät ausgeschaltet.

Auch die folgenden beschriebenen Funktionen setzen die Betriebsbereitschaft des Mobiltelefons voraus:

«PIN eingeben»
: Mit einer persönlichen Identifikationsnummer (PIN) wird die SIM-Karte und damit der Zugang zu den Mobiltelefonfunktionen geschützt. Die Eingabe einer PIN wird von den Mobiltelefonfunktionen immer dann gefordert, sobald schützenswerte Aktionen betroffen sind, wie zum Beispiel beim Einschalten des Geräts, bei der Änderung der PIN oder der PIN Kontrolle. Nach dreimaliger Falscheingabe wird die SIM-Karte gesperrt, das heißt eine PIN kann dann nicht mehr eingegeben werden[2].

«Navigieren»
: Über entsprechende Menüs erhält man Zugang zu verschiedenen Funktionen und Einstellmöglichkeiten. Die Steuerung innerhalb der Menüs wird allgemein mit der Funktion «Navigieren» bezeichnet, mit der die notwendigen Schritte zum Erreichen

[2] Durch die Eingabe einer korrekten so genannten PUK (Personal Unblocking Key-Nummer, umgangssprachlich auch Super-PIN oder Master-PIN genannt) kann die PIN Sperrung wieder aufgehoben werden. Im Rahmen dieser Arbeit wird diese Funktionalität allerdings nicht betrachtet.

einer Funktion beschrieben werden, also die gültigen Eingabesequenzen mit den entsprechenden Ausgabesequenzen.

Im Rahmen dieser Arbeit betrachten wir als Auswahlthemen exemplarisch die Sicherheits- und Tastatureinstellungen, die Ruflisten, sowie das Adressbuch und die Weckerfunktion. Das Zurücksetzen, also das unmittelbare Erreichen des Hauptmenüs im Bereitschaftszustand, ist ebenfalls ein funktionaler Bestandteil des Navigierens.

«Betriebszustand anzeigen»
: Solange keine andere Funktion im betriebsbereiten Zustand des Geräts benutzt wird, werden verschiedene Informationen, unter anderem zum Beispiel der Name des aktuellen Dienstanbieters oder Datum und Uhrzeit, auf dem Display dargestellt.

«Telefonieren»
: Die ureigenste Funktion eines Mobiltelefons ist das Telefonieren. Wir unterscheiden dabei die initiative Rolle des Benutzers mit der Teilfunktion «Nummer anrufen» und die reaktive Rolle des Benutzers mit den Teilfunktionen «Anruf annehmen» und «Anruf abweisen». Die Funktion «Anruf beenden» kann sowohl auf Initiative des Benutzers als auch vom Mobiltelefon selbst zur Anwendung kommen (wenn etwa der Gesprächspartner auflegt).

  Ein ankommendes Gespräch wird stets ungeachtet der aktuellen Nutzung anderer Funktionen angezeigt.

  «Anruf annehmen»
  : Durch Drücken einer geeigneten Taste (zum Beispiel die Abheben-Taste) wird das Gespräch angenommen. Mit der Annahme wird die mögliche Nutzung einer anderen Funktion abgebrochen.

  «Anruf abweisen»
  : Durch Drücken einer geeigneten Taste (zum Beispiel die Auflegen-Taste) wird das Gespräch abgewiesen. Die mögliche Nutzung einer anderen Funktion wird unterbrochen und nach der Abweisung fortgesetzt.

  «Anruf beenden»
  : Während des Gesprächs wird entweder durch Drücken einer geeigneten Taste (Auflegen-Taste) das Gespräch beendet oder das Mobiltelefon selbst beendet das Gespräch bei Abbrechen der Verbindung.

  «Nummer anrufen»
  : Die Eingabe einer Rufnummer mit nationaler oder internationaler Vorwahl und anschließendem Drücken einer geeigneten Taste (Abheben-Taste) wird

ein Gespräch initiiert («Nummer wählen»), welches vom Mobiltelefon in Abhängigkeit vom Zustandekommen der Verbindung entweder etabliert oder zurückgewiesen wird.

Während eines Gesprächs ist die Nutzung anderer Funktionen eingeschränkt. So werden etwa Funktionen angeboten, die nur während eines Gesprächs möglich sind (zum Beispiel «Gespräch halten»). Andere Funktionen sind dann von der Anwendung ausgeschlossen (zum Beispiel «Ausschalten») und bei wieder anderen Funktionen führt dies zu geändertem Ausgabeverhalten (zum Beispiel «Wecker»).

«Wahlwiederholung»
: Durch einmaliges Drücken der Abheben-Taste werden die vorher gewählten Rufnummern angezeigt. Die Wiederwahl einer Rufnummer erfolgt durch das Selektieren des gewünschten Eintrags und erneutes Drücken der Abheben-Taste. Da die zuletzt gewählte Rufnummer stets automatisch selektiert ist, kann durch zweimaliges Drücken der Abheben-Taste die zuletzt gewählte Rufnummer wieder gewählt werden.

«Adressen»
: Mit der Eingabe einer neuen Adresse werden Telefonnummern einem Namen zugeordnet. Die Zuordnung wird wieder aufgehoben, wenn die entsprechende Adresse wieder entfernt wird. Existiert ein Adresseintrag, so kann dieser Teilnehmer nicht nur über die Eingabe einer Telefonnummer angerufen werden, sondern auch über den zugeordneten Namen. Ebenso wird beim Verbindungsaufbau oder bei einem ankommenden Gespräch statt einer Telefonnummer der zugeordnete Name angezeigt.

«Tastensperre»
: Die Tastatur wird zum Schutz gegen unbeabsichtigtes Betätigen der Telefontasten gesperrt.

«Wecker»
: Ein Alarmton wird gespielt und im Display wird eine entsprechende Meldung angezeigt. Durch Drücken einer geeigneten Taste wird der Alarm bestätigt. Ein Gespräch wird allerdings nicht gestört und nach Beendigung des Gesprächs wird der entgangene Alarm angezeigt.

«Einstellen»
: Einige der oben angeführten Funktionen werden entsprechend des gewünschten Verhaltens konfiguriert. Unter anderem kann mit «PIN Kontrolle» der Zugriffsschutz für das Mobiltelefon über eine PIN aktiviert werden. Die Wahlwiederholung kann initialisiert werden, indem die gewählten Nummern gelöscht werden. Ebenso kann ein Weckalarm eingestellt und aktiviert werden. Mit «Jede Taste» können ankommende

Anrufe, mit Ausnahme der Auflegen-Taste, mit einer beliebigen Taste angenommen werden.

6.2 Spezifikationen als Zustandsautomaten mit Ein- und Ausgabe

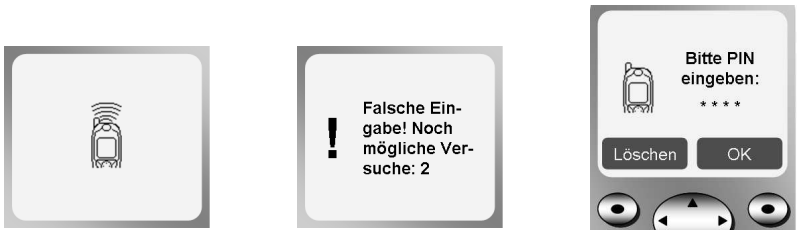
Bei den Spezifikationen der oben angeführten Funktionen wird das in Kapitel 3.4 eingeführte Transitionsschema verwendet[3]. Aus Gründen der Übersichtlichkeit werden nur die Kanäle **kbd** (Tasten), **dsp** (Anzeige) und **sys** verwendet. Da **kbd** und **dsp** klassische Eingabe- beziehungsweise Ausgabekanäle darstellen, fließen über diese Kanäle auch nur entsprechende Eingabe- und Ausgabenachrichten.

Der Verwendung der Kanalbezeichnung **sys** kommt in dieser Arbeit eine besondere Rolle zu: **sys** wird als generischer Kanal verwendet, der aus Sicht einer Funktion vom restlichen System samt seiner Menge an Funktionen abstrahiert. Für die Beschreibung einer Funktionen aus Nutzungssicht ist dies zunächst auch ausreichend. Er dient sowohl als Eingabekanal, als auch als Ausgabekanal. Über **sys** werden Nachrichten vom und zum System ausgetauscht, insbesondere kann damit auch ein interner Kanal zwischen einer nutzenden und einer genutzten Funktion modelliert werden. Im Sinne einer Verfeinerung der Funktionsgranularität (Schnittstellen) lässt sich **sys** etwa als Menge von weiteren Kanälen genauer spezifizieren (zum Beispiel als Audioschnittstelle, serielle oder Infrarot-Schnittstelle, Mobilfunknetz-Schnittstelle, und so weiter).

In den nun nachstehenden Spezifikationen der Funktionen verwenden wir für bestimmte Nachrichtentypen folgende (abkürzende) Schreibweisen.

- Nachrichten mit der Endung `_ntc` kennzeichnen eine reine Notifikation als Ausgabenachricht, zum Beispiel `logon_ntc` in Abbildung 6.2(a).
- Nachrichten mit der Endung `_dlg` kennzeichnen einen Dialog als Ausgabenachricht, welcher entsprechende Eingabenachrichten erwartet, zum Beispiel `enterpin_dlg` in Abbildung 6.2(c).

[3] Aus dem Graphviz Tookit [ATT05], einer Sammlung plattformunabhängiger Open Source Werkzeuge zum Zeichnen von Grafen, wurde *dot* [GKNV93, GKN02] für die Erstellung der Zustandsübergangsdiagramme verwendet.



(a) Einfache Notifikation (b) Parametrisierte Nachricht (c) Dialog und Soft-Keys

Abbildung 6.2: Abstrakte Nachrichten mit beispielhaften Repräsentationen im Mobiltelefon

- Bei Nachrichten mit Parametern werden die Parameter in Klammern an die Nachrichtenbezeichnung angehängt. Zum Beispiel stellt Abbildung 6.2(b) die Nachricht `fail_ntc(pin_counter)` dar, mit einem Wert von 2 für den Parameter `pin_counter`.
- Eine Nachricht in eckigen Klammern `[...]` steht als zusammengefasste Nachricht abkürzend für die Eingabe mehrerer Einzelnachrichten (zum Beispiel `[pin]` als Zusammenfassung mehrerer Ziffernnachrichten zu einer Nummernachricht).
- Nachrichten in eckigen Klammern mit dem Schlüsselwort `select` als ersten Parameter stehen abkürzend für mindestens eine Nachricht, die mit den Soft-Keys[4] des Mobiltelefons einzugeben ist, zum Beispiel `[select,OK]` in Abbildung 6.2(c). Eine Folge von Soft-Key Eingabenachrichten wird im zweiten Parameter abgekürzt als eine mit Schrägstrich (/) getrennte Folge von Nachrichten, zum Beispiel `[select,menu/setup/security]`.
- Neben den Soft-Keys spielen auch die Tasten zum Abheben und zum Auflegen eine besondere Rolle bei den Spezifikationen. Das Betätigen dieser Tasten ist dort als `off_hook` beziehungsweise `on_hook` -Nachricht auf dem Kanal **kbd** modelliert.

6.3 Von isolierten zu kombinierten Funktionsspezifikationen

Bei isolierten Funktionsspezifikationen werden zunächst bewusst keine Zusammenhänge zu anderen potenziellen Funktionen betrachtet. Solche Funktionsbeschreibungen berück-

[4] Die Belegung dieser Tasten ist dynamisch und wird entsprechend des aktuellen Zustands im Display angezeigt. Im Beispiel von Abbildung 6.2(c) bei der PIN Eingabe ist der linke Soft-Key mit „Löschen“ und der rechte Soft-Key mit „OK“ belegt.

sichtigen nur Eingabenachrichten und Ausgabenachrichten sowie funktionslokale Kontrollzustände und Datenzustände. Erst in der Kombination der Funktionen wird – als Konsequenz des Beziehungstyps – der Zusammenhang der Funktionen über die Identifizierung geeigneter gemeinsamer Daten- und Kontrollzustände hergestellt.

Ist eine Funktion unabhängig von einer anderen Funktion, so haben weder deren Kontrollzustände noch deren möglicherweise enthaltenen Vorbedingungen oder Nachbedingungen Einfluss auf das Verhalten der unabhängigen Funktion.

Nur aufgrund der Tatsache, dass zwei Funktionen Subfunktionen einer übergeordneten Superfunktion sind, kann nicht notwendigerweise auf eine Abhängigkeit der Subfunktionen geschlossen werden. Trotzdem werden oft bei der Spezifikation einer Superfunktion diejenigen Zustände der Subfunktionen vereint, die als gemeinsame Kontrollzustände oder Systemzustände identifizierbar sind.

Stehen zwei Funktionen nun aber in einem bestimmten Zusammenhang (Ursache), so ist die Spezifikation der kombinierten Funktionen im Vergleich zu den isolierten Spezifikationen geeignet zu verfeinern (Konsequenz), was zu entsprechenden unterschiedlichen Beobachtungen der Eingabehistorien und Ausgabehistorien in der Kombination führt (Wirkung). Als Konsequenzen für die kombinierte Spezifikation stehen folgende Möglichkeiten zur Verfügung:

- Einführung eines Systemzustands[5] , welcher zugleich Startzustand der einen Funktion und ein Endzustand der anderen Funktion ist.
- Einführung von Vorbedingungen bei Kontrollzuständen einer Funktion, welche sich auf Kontrollzustände der anderen Funktion beziehen.
- Einführung von Vorbedingungen bei Kontrollzuständen einer Funktion und von Nachbedingungen bei Kontrollzuständen der anderen Funktion mit gegenseitigem Bezug.
- Festlegung einer geeigneten Interaktionsschnittstelle, wobei im ersten Schritt der generische Kanal **sys** verwendet wird.

Einschalten mit Einschaltsicherung

Betrachten wir nun zunächst die Spezifikation der Funktion «Einschalten mit Einschaltsicherung».

[5] Mit einem Systemzustand wird hier ein gemeinsamer Kontrollzustand von mindestens zwei Funktionen in Kombination bezeichnet.

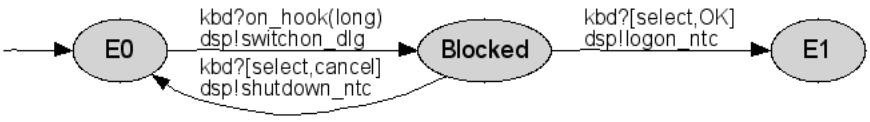


Abbildung 6.3: Einschalten des Mobiltelefons mit Einschaltsicherung (isolierte Spezifikation)

Das lange Drücken der Auflegen-Taste (auf dem Kanal **kbd** liegt die Nachricht `on_hook(long)` an) führt zu einer abstrakten Ausgabenachricht, und zwar zum Bestätigungsdialog für das Einschalten des Geräts (auf dem Kanal **dsp** ist die Nachricht `switchon_dlg`) und den Übergang in den Zustand *Blocked*. Dann kann durch die Wahl der Abbruch-Option (auf dem Kanal **kbd** liegt die Nachricht `[select,cancel]` an) der Einschaltvorgang verhindert werden und das Gerät wird wieder im Ausgangszustand *E0* sein. Wird über `[select,OK]` das Einschalten bestätigt, erfolgt die Ausgabe einer Anmeldenachricht sowie der Übergang in den Zustand *E1*.

Einschalten und Ausschalten kombiniert

Eine weitere Nutzungsfunktion ist das «Ausschalten» des Geräts. Diese Funktion sei wie folgt spezifiziert:

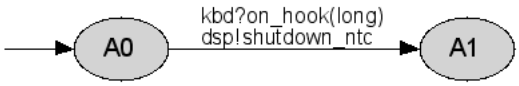


Abbildung 6.4: Ausschalten des Mobiltelefons (isolierte Spezifikation)

Auch hier führt aus dem Anfangszustand *A0* das lange Drücken der Auflegen-Taste (`on_hook(long)` auf dem Kanal **kbd**) zum Übergang in einen anderen Zustand *A1*, allerdings wird dabei eine andere Nachricht ausgegeben, und zwar die Abmeldenachricht `shutdown_ntc`.

Die uneingeschränkte, beliebig verschränkte Nutzung dieser beiden Funktionen ist offensichtlich nicht möglich. Beide Funktionen besitzen eine gemeinsame Eingabe (hier nur die eine Nachricht `on_hook(long)`) und zugleich sind dafür unterschiedliche Ausgaben spezifiziert. Dies sind offenbar widersprüchliche Anforderungen für die Ausgabehistorien, was – ohne Treffen geeigneter Maßnahmen – zunächst die Nicht-Kombinierbarkeit dieser Funktionen bedeutet. Eine Maßnahme ist nun, die Beziehung der Funktionen zu analysieren. Die Funktionen können beide trotzdem zugleich Subfunktionen des Mobiltelefons

sein, wenn etwa die simultane Nutzung verhindert wird. Wir drücken dies aus durch den Beziehungstyp exclude.

'*Einschalten mit Einschaltsicherung*' **excludes** '*Ausschalten*'

Die Forcierung der sequenziellen Nutzung der Funktionen ist leicht zu erreichen, wenn der Anfangszustand der einen Funktion mit einem Endzustand der anderen Funktion übereinstimmt. Es liegt nun nahe, dass das Gerät erst ausgeschaltet werden kann, wenn es vorher eingeschaltet wurde.

'*Einschalten mit Einschaltsicherung*' **prepares** '*Ausschalten*'

Als Konsequenz vereinen wir dazu nun den Endzustand von «Einschalten» mit dem Anfangszustand von «Ausschalten» und führen die Zustände *E1* und *A0* zusammen in dem übergreifenden Kontrollzustand *Standby*. Darüber hinaus unterstellen wir noch die Beziehung

'*Ausschalten*' **prepares** '*Einschalten mit Einschaltsicherung*'

da das Gerät selbstverständlich auch nur dann eingeschaltet werden kann, wenn es vorher ausgeschaltet wurde (mit Ausnahme des Spezialfalls des ersten Einschaltens). In dem übergreifenden Kontrollzustand *Off* können damit auch die Zustände *A1* und *E0* zusammengeführt werden. Man erhält somit als Spezifikation der kombinierten Funktionen das Zustandsübergangsdiagramm in Abbildung 6.5.

Einschalten mit Zugriffsschutz

Neben der Einschaltsicherung existiert außerdem die üblichere Variante des Einschaltvorgangs, und zwar die des Zugriffsschutzes mit PIN. Wir verwenden hier ebenfalls die bereits eingeführten Kontrollzustände *Off* und *Standby* und gehen damit von einer Spezifikation aus, wie sie in Abbildung 6.6 dargestellt ist.

Wie vorher, führt auch hier das lange Drücken der Auflegen-Taste (`on_hook(long)` auf dem Kanal **kbd**) vom Startzustand in einen anderen Zustand, der hier allerdings mit *Locked* bezeichnet ist. Ein weiterer Unterschied ist dabei die Ausgabenachricht `pin_req` auf dem Kanal **sys**. Wie bereits erwähnt, verwenden wir die Kanalbezeichnung **sys** allgemein für eine noch zu verfeinernde Schnittstelle. Hier wird eine weitere Funktion des Systems genutzt, und zwar derart, dass das weitere Verhalten abhängig vom gelieferten Ergebnis der genutzten Funktion ist. Dies wird ausgedrückt durch den Beziehungstyp consult.

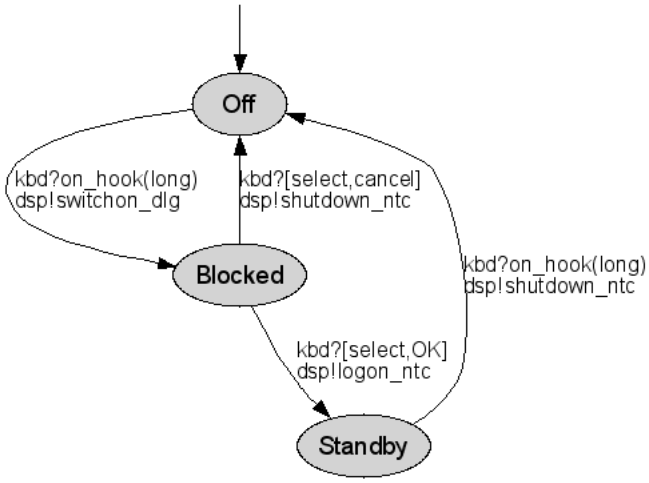


Abbildung 6.5: Ein- und Ausschalten des Mobiltelefons mit Einschaltsicherung

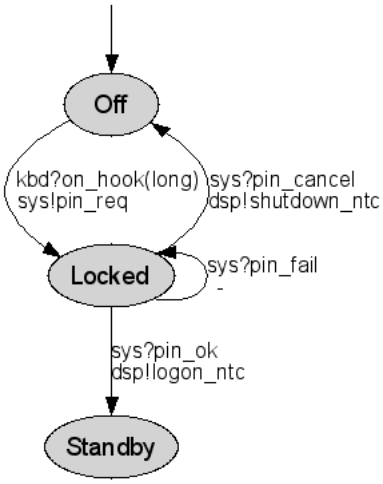


Abbildung 6.6: Einschalten des Mobiltelefons mit Zugriffsschutz (isolierte Spezifikation)

Benutzung der PIN Eingabe beim Einschalten mit Zugriffsschutz

Die Nachricht `pin_req` veranlasst hier also die Nutzung einer weiteren Funktion des Systems, wobei im Zustand *Locked* mögliche Ergebnisse der benutzten Funktion verarbeitet werden. Die spezifizierten erwarteten Nachrichten sind `pin_ok`, `pin_fail` und `pin_cancel`, welche ebenfalls über den generischen Kanal **sys** gelesen werden. Diese benutzte Funktion ist hier «PIN eingeben» (siehe auch Abbildung 6.8 auf Seite 146), wobei es an dieser Stelle allerdings genügt, die Eingabe- und Ausgabenachrichten dieser Funktion zu kennen,

> '*Einschalten mit Zugriffsschutz*' **consults** '*PIN eingeben*'.

Der Abbruch der PIN Eingabe (`pin_cancel` auf dem Kanal **sys**) verhindert analog wie bei der Eingabe mit Einschaltsicherung den Einschaltvorgang und das Gerät befindet sich wieder im Zustand *Off*. Bei erfolgreicher PIN Eingabe (`pin_ok` auf dem Kanal **sys**) wird der Einschaltvorgang vollendet, auch hier mit der Ausgabe einer Anmeldenachricht sowie dem Übergang in den Zustand *Standby*.

Einschalten mit Einschaltsicherung und mit Zugriffsschutz in Kombination

Wie oben bereits dargestellt werden die Zustände *Off* und *Standby* bei den beiden Funktionen «Einschalten mit Einschaltsicherung» und «Einschalten mit Zugriffsschutz» als gemeinsame übergreifende Kontrollzustände benützt. Es fällt allerdings auf, dass bei den angegebenen Transitionen aus dem Startzustand *Off* heraus die Eingabenachricht `on_hook(long)` jeweils unterschiedliches Ausgabeverhalten nach sich zieht. Auch dies sind zunächst widersprüchliche Ausgaben, wie wir ihn vorher im Zusammenhang mit der exclude Beziehung zur «Ausschalten» Funktion kennen gelernt haben. In der Kombination resultiert dies in der Funktion «Einschalten» des Mobiltelefons als Superfunktion der beiden Funktionen, bei deren Spezifikation ebenfalls die gemeinsamen Zustände verwendet werden. Wir schreiben

> **subfunction**(*Einschalten*, *Einschalten mit Einschaltsicherung*) und
> **subfunction**(*Einschalten*, *Einschalten mit Zugriffsschutz*).

Die isolierten Spezifikationen der beiden Subfunktionen lassen sich hier leicht wieder erkennen. Man beachte jedoch, dass bei dieser Spezifikation der «Einschalten» Funktion die Wahl der alternativen Einschaltmechanismen und damit das Verhalten der Funktion nicht deterministisch ist. Aus dem Zustand *Off* kann durch die `on_hook(long)` Nachricht auf dem Kanal **kbd** sowohl der Zustand *Blocked* als auch der Zustand *Locked* erreicht werden.

Um hier den Widerspruch aufzulösen, kann etwa der Funktion «Einschalten mit Zugriffs-

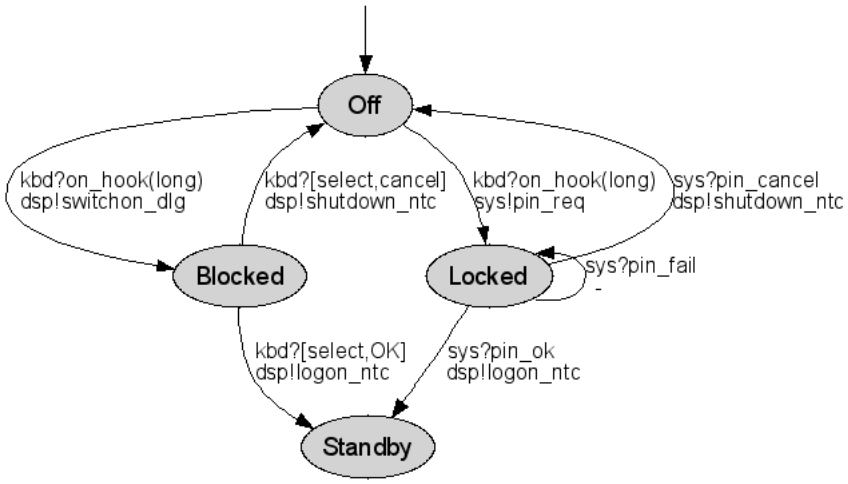


Abbildung 6.7: Einschalten mit Einschaltsicherung oder Zugriffsschutz

schutz» eine höhere Priorität beigemessen werden.

'*Einschalten mit Zugriffsschutz*' **replaces** '*Einschalten mit Einschaltsicherung*'

Ein andere Lösung besteht darin, dass eine weitere Funktion die Funktionen jeweils über geeignete Bedingungen zugleich aktiviert beziehungsweise deaktiviert. Wie wir sehen, ist dies die in der Fallstudie gewählte Variante mit der Funktion «PIN Kontrolle».

PIN eingeben

Die Funktion «PIN eingeben» ist eine weitere Subfunktion des Mobiltelefons, auf die bereits in der Kombination beim Einschalten mit Zugriffsschutz hingewiesen wurde. Im Rahmen dieser Fallstudie wird sie ebenfalls von der Funktion «PIN Kontrolle» benutzt. «PIN eingeben» ist ein typischer Repräsentant von Funktionen, welche nur über die Nutzung durch andere Funktionen des Systems zu beobachten sind. Die Domänen solcher Funktionen beinhalten nur Eingabehistorien, die mit Nachrichten auf internen Kanälen beginnen. In dem Fall beginnt jede gültige Eingabehistorie mit der Nachricht `pin_req` auf dem Kanal **sys**.

Die in Abbildung 6.8 angegebenen Subfunktionen sind nicht mehr so einfach zu erkennen wie etwa oben bei der Kombination der Einschaltvarianten in Abbildung 6.7. Jede isolierte Spezifikation der drei Subfunktionen beinhaltet alle fünf dargestellten Kontrollzustände. Auch das Verhalten der Funktionen ist bis auf die letzte Nachricht identisch.

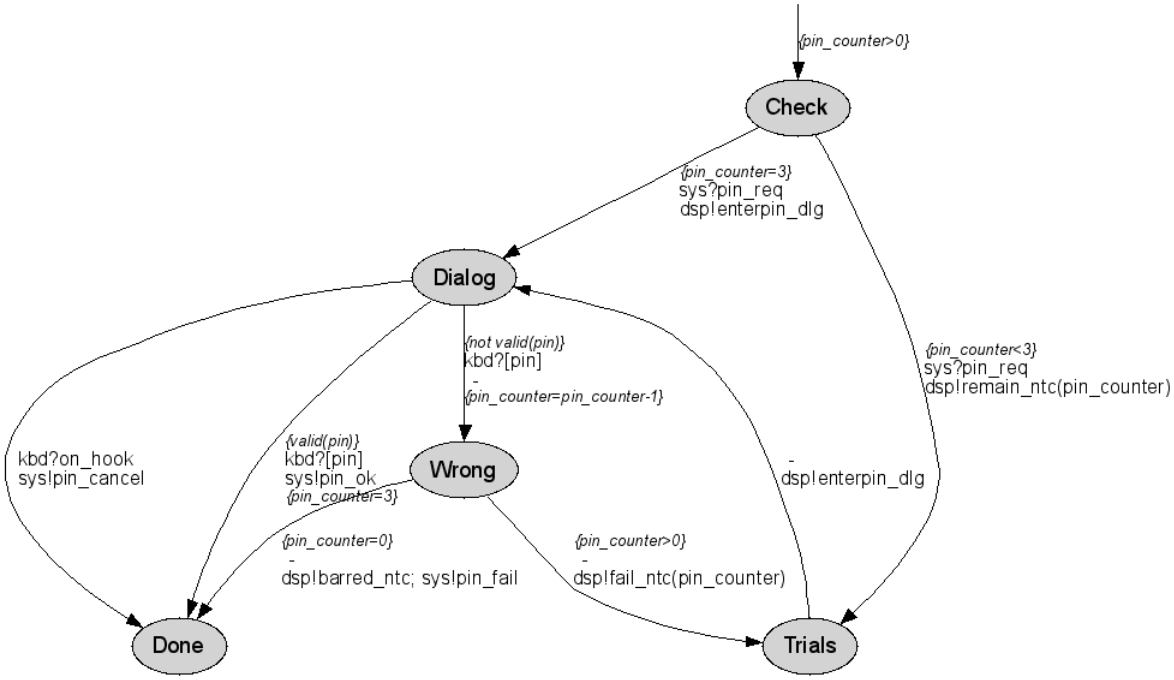


Abbildung 6.8: PIN eingeben mit den Subfunktionen «Korrekte PIN eingeben», «Falsche PIN eingeben», sowie «PIN Eingabe abbrechen»

Sie unterscheiden sich nur durch einen jeweils unterschiedlichen Übergang in den Zustand *Done*.

	Suffix der Eingabeströme auf dem Kanal **kbd**	Suffix der Ausgabeströme auf dem Kanal **sys**
«PIN Eingabe abbrechen»	`<..., on_hook>`	`<..., pin_cancel>`
«Korrekte PIN eingeben»	`<..., [pin]>`	`<..., pin_ok>`
«Falsche PIN eingeben»	`<..., [pin]>`	`<..., pin_fail>`

Die in der Tabelle punktiert angedeuteten Teilnachrichtenströme (`...`) sind für alle drei Subfunktionen identisch. Nach höchstens zwei Fehlversuchen kann noch eine PIN korrekt eingegeben werden oder die PIN Eingabe abgebrochen werden. Mit der Funktion «Falsche PIN eingeben» wird das Verhalten einer dreimaligen Falscheingabe der PIN beschrieben.

Wie sich an den Vorbedingungen der Transitionen zu den Kontrollzuständen *Dialog* und *Trials* ablesen lässt, ist die Funktion «PIN eingeben» partiell definiert, und zwar genau

dann, wenn `pin_counter` die Werte 1, 2 oder 3 annimmt. Bei korrekter PIN Eingabe wird `pin_counter` stets auf den Wert 3 gesetzt. Bei der nächsten Benutzung der Funktion sind dann wieder insgesamt drei Versuche zur PIN Eingabe – also zwei Fehlversuche – möglich. Bei Abbruch der PIN Eingabe bleibt dagegen die Anzahl der möglichen PIN Eingaben unverändert.

Es ist bemerkenswert, dass der Wert von `pin_counter` in der Funktion selbst, und zwar in den Nachbedingungen der Transitionen aus dem Zustand *Dialog* heraus, verändert wird. In diesem Sinne beeinflusst sich die Funktion gewissermaßen selbst. Dies ist an sich noch nicht außergewöhnlich, allerdings wird durch das Eingeben der falschen PIN insbesondere `pin_counter` auf den Wert 0 gesetzt. Das Verhalten der Funktion ist für diese Belegung jedoch nicht definiert. Es gilt

‘*Falsche PIN eingeben*’ **disables** ‘*PIN eingeben*’.

Hier erwartete man zum Beispiel die Steuerung durch eine weitere Funktion, etwa «PUK eingeben» mit der Beziehung

‘*PUK eingeben*’ **enables** ‘*PIN eingeben*’,

welche die Annahmen gemäß der Spezifikation wieder herstellt. Eine derartige Funktion entspricht sinngemäß in etwa der PIN Eingabe und wird deshalb nicht mehr im Rahmen dieser Fallstudie behandelt[6].

Der Kontrollzustand *Done* bei einer benutzten Funktion kennzeichnet gleichsam die Rückkehr zum Kontrollfluss der aufrufenden Funktion (consult Beziehung). Aus diesem Zustand wird kein anderer Kontrollzustand der Funktionsspezifikation erreicht. Die Transition in diesen Endzustand enthält allerdings entsprechende Ausgabenachrichten, welche das Verhalten der aufrufenden Funktion steuert, hier `pin_cancel`, `pin_ok`, `pin_fail` im Kontrollzustand *Locked*.

Eine benutzte Funktion muss nicht zwangsläufig für den Benutzer unsichtbar sein, nur weil sie über interne Kanäle mit der nutzenden Funktion komponiert wird. Sie hat in der Tat auch von außen beobachtbar im Sinne zusätzlicher Interaktionen zu sein. Dies ist auch hier der Fall: Der Kanal **sys** dient zwar als interne Schnittstelle zwischen den Funktionen, zum Beispiel zwischen «Einschalten mit Zugriffsschutz» und «PIN eingeben». Aber «PIN eingeben» selbst definiert wiederum Interaktionen mit dem Benutzer über die Kanäle **kbd** und **dsp**. Aus Nutzersicht werden hierbei keine voneinander losgelösten Interaktionsmuster wahrgenommen, sondern als ein integriertes Muster, welches beide spezifizierten Ein- und Ausgaben beinhaltet.

[6] Siehe auch den kurzen Abriss der Funktion «PIN eingeben» in Kapitel 6.1 auf Seite 135.

PIN Kontrolle

Die Funktion «Einstellen» bietet eine Reihe unterschiedlichster Subfunktionen, mit denen typischerweise andere Funktionen konfiguriert werden. Unter anderem kann mit der Funktion «PIN Kontrolle» der Zugriffsschutz für das Mobiltelefon über eine PIN aktiviert, aber auch deaktiviert werden,

subfunction(*Einstellen*, *PIN Kontrolle*).

In der folgenden Spezifikation der Funktion fällt zunächst auf, dass als Startzustand wieder der Kontrollzustand *Standby* gewählt wurde. Es gilt also die Beziehung

'*Einschalten*' **prepares** '*PIN Kontrolle*'.

Hier kommt der prepare Beziehungstyp allerdings nicht deshalb zur Anwendung, weil es Eingabekonflikte zu vermeiden gilt (vergleiche die Kombination der Funktionen «Einschalten» und «Ausschalten» auf Seite 141). Eine weitere Verwendung ist die Umsetzung unterschiedlicher Systemmodi. In Abhängigkeit vom Systemmodus soll nur eine bestimmte Menge von Funktionen zur Verfügung stehen. In diesem Falle steht die Funktion «PIN Kontrolle» nur dann zur Verfügung, wenn das Gerät betriebsbereit und insbesondere eingeschaltet ist.

Der Zustandsübergang von Standby heraus weist eine abkürzende Schreibweise zur Erreichung des Zustands *Security Setup* auf. Die Nachricht `[select,menu/setup/security]` abstrahiert gewissermaßen die notwendigen Teile der Eingabehistorien. Im Zusammenhang mit der Funktion «Navigieren» wird dies nachfolgend ausführlich besprochen. Die Interaktion der «Navigieren» Funktion wird durch zusätzliche Eingabehistorien (und Ausgabehistorien) von «PIN Kontrolle» ergänzt,

'*PIN Kontrolle*' **complements** '*Navigieren*'.

In der Spezifikation der Funktion zeigt sich unter anderem ebenfalls die Nutzung der Funktion «PIN eingeben» mit den entsprechenden Nachrichten über den Kanal **sys**,

'*PIN Kontrolle*' **consults** '*PIN eingeben*'.

Dies wurde bereits im Falle des Einschaltens mit Zugriffsschutz besprochen. Mit Ausnahme des festgelegten Kontrollzustands, hier *PIN Check*, zur Verarbeitung möglicher Ergebnisse der benutzten Funktion, ist die Spezifikation an der Stelle analog.

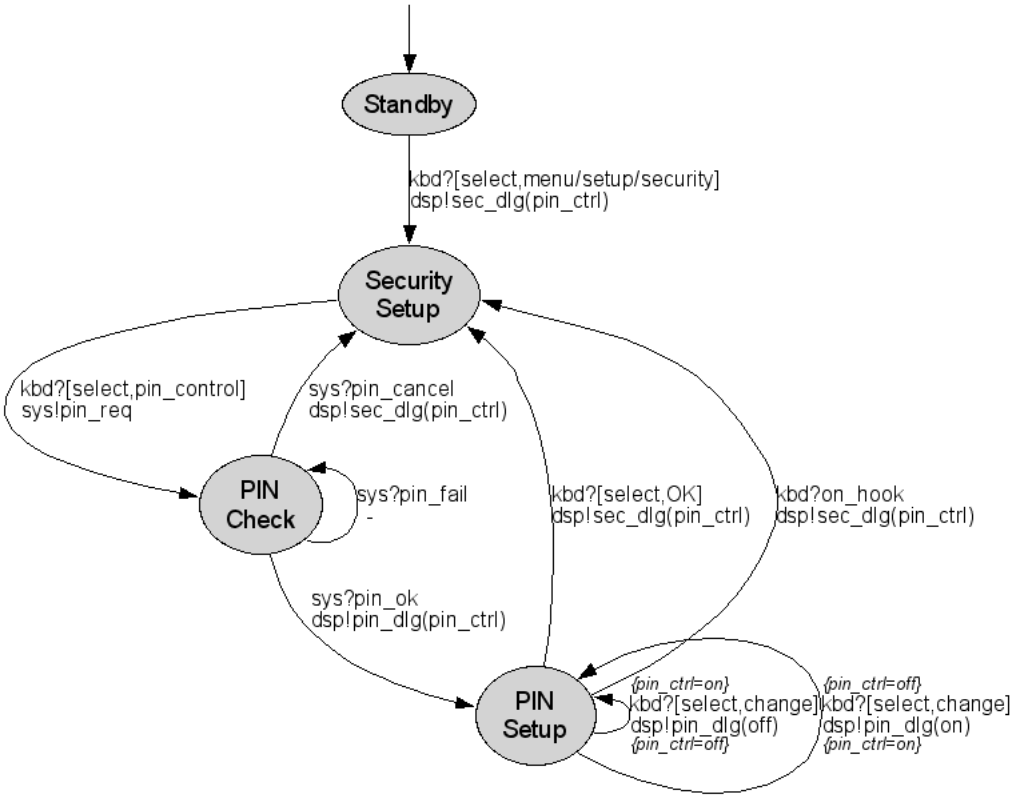


Abbildung 6.9: PIN Kontrolle (Aktivierung und Deaktivierung) mit PIN Prüfung

Bei erfolgreicher PIN Eingabe kann nun zwischen der Aktivierung und der Deaktivierung der PIN Kontrolle gewechselt werden. Mit anderen Worten: Je nach Auswahl wird der Datenzustand für `pin_ctrl` entweder in `On` oder `Off` geändert. Mit diesem Schalter lässt sich nun die Spezifikation für das «Einschalten» deterministisch gestalten. In Abhängigkeit des `pin_ctrl` Zustands steht damit beim Einschalten nur die entsprechende Subfunktion zur Verfügung. Es gilt zum Beispiel:

'*PIN Kontrolle*' **enables** '*Einschalten mit Zugriffsschutz*' und
'*PIN Kontrolle*' **disables** '*Einschalten mit Einschaltsicherung*'.

Einschalten mit Einschaltsicherung und mit Zugriffsschutz – deterministisch kombiniert

Das Ein- und Ausschalten des Geräts kann nun vollständig definiert werden (siehe Abbildung 6.10). Mit den entsprechenden Gates als Vorbedingungen für die Transitionen, die in «PIN Kontrolle» festgelegt werden, ist die Wahl der Einschaltmechanismen eindeutig. Aus dem Zustand *Off* wird durch die `on_hook(long)` Nachricht auf dem Kanal **kbd** entweder der Zustand *Blocked* erreicht, falls die PIN Kontrolle deaktiviert ist (`pin_ctrl=off`), oder der Zustand *Locked* bei aktivierter PIN Kontrolle (`pin_ctrl=on`).

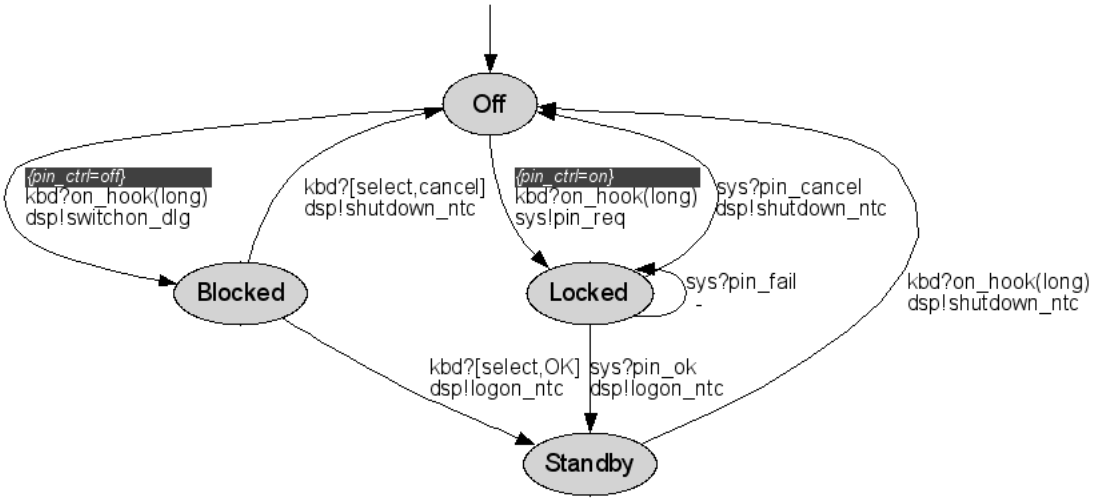


Abbildung 6.10: Ein- und Ausschalten des Mobiltelefons mit Einschaltsicherung beziehungsweise mit Zugriffsschutz

In Abbildung 6.10 sind die Übergänge zwischen den Betriebsmodi *Off* und *Standby* dargestellt, mit den kombinierten Subfunktionen «Einschalten», «Ausschalten» und «PIN Kontrolle». Im Übrigen ist «Einschalten» in Kombination mit «PIN Kontrolle» nach wie vor eine Superfunktion von «Einschalten mit Einschaltsicherung» und «Einschalten mit Zugriffsschutz». Die Einführung von Vorbedingungen zur Selektion der Subfunktionen führt nur zu einer eingeschränkten Subfunktionsrelation beziehungsweise zu einer bedingten, also an bestimmte Bedingungen geknüpfte Kombinierbarkeit der Funktionen. Diese Bedingungen können durch die Angabe von Teilmengen der Domäne von «PIN Kontrolle» formuliert werden. Wir wollen diesen speziellen Zusammenhang hier am Beispiel nachvollziehen und stellen dazu folgende Überlegung an:

Eine die beiden Funktionen «Einschalten mit Zugriffsschutz» (nachfolgend abgekürzt mit „EmZ“) und «PIN Kontrolle» umfassende Superfunktion ist in dieser Fallstudie die Funktion «Mobiltelefon» selbst. Die Menge der gültigen Eingaben für «PIN Kontrolle» wird mit

$$P_V^{Mobiltelefon}(PIN\ \ Kontrolle) \\ = \{x \in \mathsf{dom}(Mobiltelefon) : (x \,|\, I_{PIN\ \ Kontrolle}) \in \mathsf{dom}(PIN\ \ Kontrolle)\}$$

angegeben. Sei mit $E_{Act} = P_{ENA}^{Mobiltelefon}(EmZ)$ nun die Menge der gültigen Eingabehistorien zur Aktivierung der PIN Kontrolle bezeichnet.

$$E_{Act} \subset P_V^{Mobiltelefon}(PIN\ \ Kontrolle)$$

Bezogen auf E_{Act} ist die Subfunktionsrelation für «Mobiltelefon» und «Einschalten mit Zugriffsschutz» gegeben. Wir schreiben

$$\mathbf{subfunction}(Mobiltelefon \,|\, E_{Act}, EmZ)$$

denn es gilt nur unter der Bedingung E_{Act}:

$$\forall\, x \in \mathsf{dom}(EmZ) : Mobiltelefon\dagger(I_{EmZ} \blacktriangleright O_{EmZ}).x = EmZ.x \\ \wedge\ \ \mathsf{dom}(EmZ) = \mathsf{dom}(Mobiltelefon\dagger(I_{EmZ} \blacktriangleright O_{EmZ}))$$

Die Subfunktionsbeziehung gilt deshalb nicht uneingeschränkt, da im Falle der deaktivierten PIN Kontrolle hier die vorgegebenen Eingabehistorien von «Einschalten mit Zugriffsschutz» nicht definiert sind.

$$\exists\, x \in \mathsf{dom}(Mobiltelefon) : \\ (x \,|\, I_{PIN\ \ Kontrolle}) \notin E_{Act}\ \ \wedge\ \ (Mobiltelefon.x) \,|\, O_{EmZ} = \varnothing$$

Nebenbei bemerkt gilt die eingeschränkte Subfunktionsrelation auch für die Funktion «Einschalten mit Einschaltsicherung». Die Bedingung ist dort allerdings an die gültigen Eingabehistorien zur Deaktivierung der PIN Kontrolle geknüpft:

$$\mathbf{subfunction}(Mobiltelefon \setminus E_{Act}, Einschalten\ \ mit\ \ Einschaltsicherung).$$

Mehrfache Beziehungen zwischen zwei Funktionen

Die oben beschriebenen enable- und disable- Beziehungen der «PIN Kontrolle» Funktion sind im Grunde noch nicht vollständig. Es gilt zusätzlich:

‘*PIN Kontrolle*’ **enables** ‘*Einschalten mit Einschaltsicherung*’ und
‘*PIN Kontrolle*’ **disables** ‘*Einschalten mit Zugriffsschutz*’.

Beispielsweise steht also «PIN Kontrolle» und «Einschalten mit Zugriffsschutz» sowohl in einer enable Beziehung als auch in einer disable Beziehung. Das ist auch zweifelsohne nicht falsch, allerdings wird mit mehrfach definierten Beziehungen die Nachvollziehbarkeit des Verhaltens in der Kombination der Funktionen nicht gerade einfacher. Eine derartige Situation ist oft bei Funktionen gegeben, welche sich prinzipiell weiter dekombinieren ließen, und zwar als Kombination von Subfunktionen. Mit anderen Worten: durch die Dekombination einer Funktion können sich mehrfache Beziehungen umgehen lassen. In diesem Beispiel kann die Funktion mit

subfunction(*PIN Kontrolle*, *PIN Kontrolle aktivieren*) und
subfunction(*PIN Kontrolle*, *PIN Kontrolle deaktivieren*)

dekombiniert werden, wobei dann folgenden Beziehungen gelten:

'*PIN Kontrolle aktivieren*' **enables** '*Einschalten mit Zugriffsschutz*' und
'*PIN Kontrolle aktivieren*' **disables** '*Einschalten mit Einschaltsicherung*'

sowie

'*PIN Kontrolle deaktivieren*' **enables** '*Einschalten mit Einschaltsicherung*' und
'*PIN Kontrolle deaktivieren*' **disables** '*Einschalten mit Zugriffsschutz*'.

Gemäß der Spezifikation von der Funktion «Einschalten» werden deren Subfunktionen über Vorbedingungen differenziert. In diesem speziellen Fall eignet sich auch der Beziehungstyp direct, den Zusammenhang zu beschreiben,

'*PIN Kontrolle*' **directs** '*Einschalten*'

denn die zur Anwendung kommende Teilmenge der Eingabehistorien von «Einschalten» wird von der Funktion «PIN Kontrolle» bestimmt.

Das Navigieren

Im Zusammenhang mit der Beschreibung der Funktion «PIN Kontrolle» auf Seite 148 wurde bereits auf die «Navigieren» Funktion verwiesen. Dort wurde der Zustandsübergang von *Standby* nach *Security Setup* durch geeignete Eingabenachrichten respektive eine Reihe geeigneter Eingaben am Gerät erreicht, welche im Zustandsübergangsdiagramm abkürzend mit `menu/setup/security` beschrieben waren. «Navigieren» bietet allgemein die Funktionalität, aus dem Bereitschaftszustand heraus auf verschiedene Funktionen zuzugreifen und auch wieder in den Bereitschaftszustand zu gelangen. Es gilt

'*Einschalten*' **prepares** '*Navigieren*'.

In Abbildung 6.11 ist der Bereitschaftszustand wie vorher mit *Standby* markiert. Durch Drücken des „Menü“ Soft-Keys (siehe nebenstehende Grafik) wird das Navigieren ausgelöst. In der oben erwähnten Folge der Soft-Key Eingaben `menu/setup/security` wird dies durch das erste Element der Folge, `menu`, umschrieben. Das nächste Element `setup` umschreibt eine weitere Soft-Key Eingabe, mit der der Zustandsübergang nach *Sub Menu* vollzogen wird, wenn wir `[select,'submenu']` durch `[select,setup]` substituieren. Alle weiteren Elemente der Eingabefolge verändern lediglich die Menütiefe.

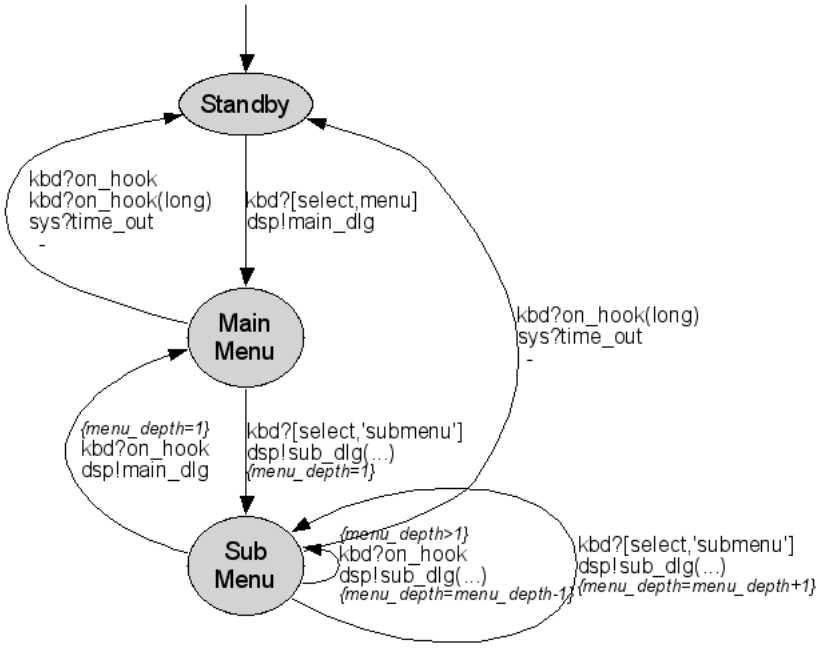


Abbildung 6.11: Navigieren

Durch kurzes Drücken der Auflegen-Taste (`on_hook` auf dem Kanal **kbd**) wird stets eine Ebene zum übergeordneten Menü zurückgesprungen. Die Subfunktion «Rücksetzen» ermöglicht das Erreichen des Bereitschaftszustands aus einer beliebig erreichten Menütiefe heraus. Der Zustand *Standby* wird dabei durch langes Drücken der Auflegen-Taste (`on_hook(long)` auf dem Kanal **kbd**) oder durch ein Signal des Geräts selbst (`time_out` auf dem Kanal **sys**) erreicht.

Die Interaktion der «Navigieren» Funktion wird üblicherweise durch andere Funktionen

ergänzt, zum Beispiel gilt wie bereits erwähnt

‘*PIN Kontrolle*’ **complements** ‘*Navigieren*’.

Im Vorgriff auf den nächsten Abschnitt gelten auch darüber hinaus die folgenden Beziehungen:

‘*Wahlwiederholung*’ **complements** ‘*Navigieren*’,
‘*Jede Taste*’ **complements** ‘*Navigieren*’,
‘*Adressen*’ **complements** ‘*Navigieren*’ und
‘*Wecker aktivieren*’ **complements** ‘*Navigieren*’.

Alle Spezifikationen dieser Funktionen kennzeichnen mit der abkürzenden Schreibweise für die Folge von Eingabenachrichten gleichsam die Stelle, an der die Erweiterung sichtbar wird. Die Domäne von «Mobiltelefon» enthält Eingabehistorien, welche, projiziert auf die Schnittstelle der «Navigieren» Funktion, die beschriebenen Eingaben von «Navigieren» enthalten. Zusätzlich beinhaltet sie Eingabehistorien, die sich aus den gültigen Eingaben von «Navigieren» und den ergänzenden Funktionen zusammensetzen.

Nicht zuletzt sei noch angemerkt, dass die Folgen von Menüschritten selbstverständlich gerätespezifisch sind und sollten deshalb nur exemplarisch betrachtet werden. Die Spezifikation kann allerdings leicht unter Beibehaltung des Prinzips des Navigierens durch Ersetzen der Menüsequenz entsprechend angepasst werden.

6.4 Weitere Funktionen und Beziehungen

Im vorigen Kapitel wurde anhand der Funktionen «Einschalten», «Ausschalten», «PIN eingeben», «PIN Kontrolle» und «Navigieren» die systematische Entwicklung kombinierter Spezifikationen auf der Grundlage von Funktionsbeziehungen gezeigt. Dabei kamen bereits die Beziehungstypen subfunction, exclude beziehungsweise prepare, sowie consult, enable, disable, direct und complement zur Anwendung. Im Folgenden werden wir nun weniger auf die Details der Funktionen und deren Kombinationen eingehen, sondern vielmehr auf weitere Beziehungen zwischen Funktionen, welche insbesondere die restlichen in Kapitel 5 definierten Beziehungstypen rechtfertigen.

Grundlegende Funktionen des Telefonierens

Mit den hier ausgewählten Subfunktionen der Funktion «Telefonieren» lässt sich eine Reihe weiterer Beziehungen veranschaulichen, die auch neue Beziehungstypen erfordern. Die Komplexität des Zustandsübergangsdiagramms in Abbildung 6.12 ist vergleichsweise

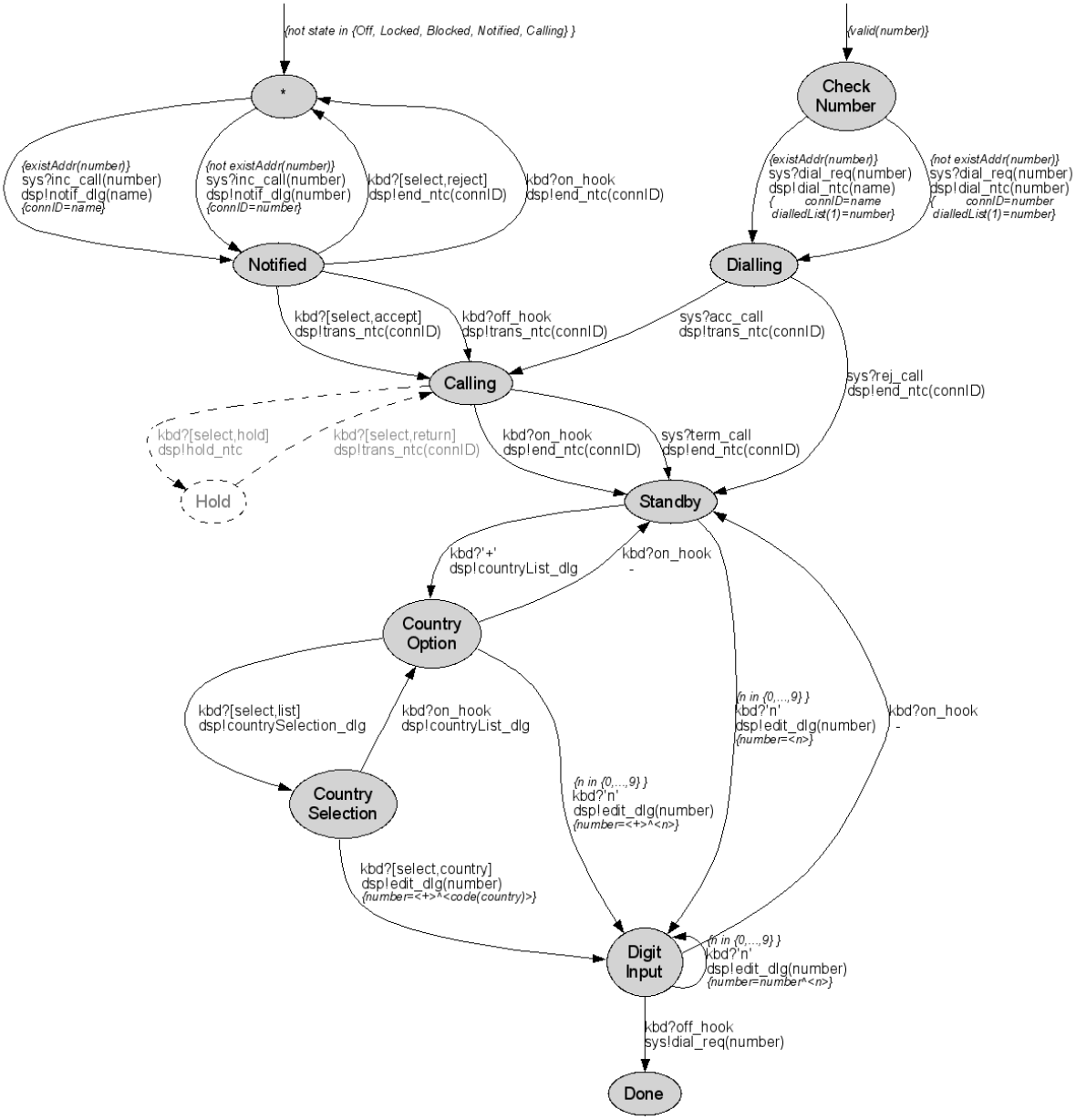


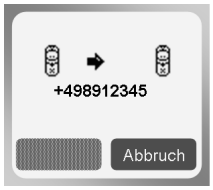
Abbildung 6.12: Telefonieren mit den Subfunktionen «Anruf annehmen», «Anruf abweisen», «Anruf beenden», «Nummer anrufen» mit internationaler Vorwahl sowie «Nummer wählen»

höher als bei den vorhergehenden Funktionen, da es bereits die Spezifikationen aller Subfunktionen in Kombination beinhaltet.

Die ersten beiden Funktionen, die wir betrachten möchten, haben ihren Ausgangspunkt in dem Kontrollzustand *Standby*. Die Funktion «Nationale Nummer» umfasst die Kontrollzustände [*Standy − Digit Input − Done*]. Im Bereitschaftszustand des Geräts kann über die Tasten eine Telefonnummer eingegeben werden. Diese Eingabe kann dann entweder abgebrochen werden oder die Nummer gewählt werden. Dies wird durch die Transitionen [*Digit Input − Standby*] (`on_hook` auf dem Kanal **kbd**) beziehungsweise [*Digit Input − Done*] (`off_hook` auf dem Kanal **kbd**) umgesetzt.

Die Eingabe eines führenden Plus Zeichens (+) initiiert die Funktion «Internationale Vorwahl», welche die Transitionen [*Standby − Country Option − Country Selection − Digit Input*] umfasst. Eine internationale Vorwahl kann sowohl direkt eingegeben als auch ausgewählt werden. Durch Anfügen der entsprechenden nationalen Rufnummer erhält man die vollständige Telefonnummer. Die Domäne der Superfunktion «Nummer anrufen» beinhaltet neben den Eingabehistorien der «Nationale Nummer» Funktion zusätzlich Eingabehistorien, die sich aus den Eingabehistorien der «Nationale Nummer» Domäne und der «Internationale Vorwahl» Domäne zusammensetzen. Es gilt

‘*Internationale Vorwahl*’ **complements** ‘*Nationale Nummer*’.



Die Funktion «Nummer anrufen» besitzt noch eine weitere Subfunktion. Die Funktion «Nummer wählen», welche die Transitionen [*Check Number − Dialling − Calling/Standby*] einschließt, wird typischerweise von anderen Funktionen genutzt, indem der Kontrollfluss übergeben wird. Dies zeigt sich an den Eingabenachrichten über den Kanal **sys**, wie sie auch vorher im Zusammenhang mit einer consult Beziehung aufgetaucht sind. Es sind allerdings keine Ausgabenachrichten über den Kanal **sys** definiert, welche das Verhalten der nutzenden Funktion beeinflusst. Die Subfunktion «Nationale Nummer» benutzt nun «Nummer wählen» genau über diesen Mechanismus, das durch die Beziehung

‘*Nationale Nummer*’ **triggers** ‘*Nummer wählen*’

formuliert wird.

Der Kontrollzustand *Calling* repräsentiert den Kontext eines laufenden Telefongesprächs. Dieser Zustand wird im Falle der Funktion «Nummer wählen» erreicht, wenn der Gesprächspartner den Anruf angenommen hat (`acc_call` auf dem Kanal **sys**). Bei einer Abweisung des Anrufs wird das Gerät dagegen in den Bereitschaftszustand versetzt.

Im Kontext eines Telefonats wird üblicherweise eine Vielzahl gesprächsspezifischer Funktionen angeboten, auf die wir hier nicht näher eingehen. In Abbildung 6.12 ist dies lediglich durch gestrichelt markierte Transitionen angedeutet. Eine spezielle Funktion sei dennoch herausgestellt, und zwar «Anruf beenden» mit den Transitionen [*Calling* − *Standby*].

Für das Beenden eines Gesprächs sind hier zwei Varianten vorgesehen: das aktive Auflegen (`on_hook` auf dem Kanal **kbd**) oder das Gespräch wird beendet (`term_call` auf dem Kanal **sys**), wenn zum Beispiel der Gesprächspartner aufgelegt hat. Die Funktion «Anruf beenden» kann nur in einem bestimmten Systemmodus zur Verfügung stehen, welcher mit dem Kontrollzustand *Calling* gekennzeichnet ist. Nachdem unter anderem über die Funktion «Nummer wählen» dieser Zustand erreicht wird, gilt:

‘*Nummer wählen*’ **prepares** ‘*Anruf beenden*’.

Als letzte Subfunktionen im Rahmen der «Telefonieren» Funktion diskutieren wir die Funktionen «Anruf annehmen» und «Anruf abweisen». Die Transitionen [* − *Notified* − *] umfassen die Funktion «Anruf abweisen», während «Anruf annehmen» mit den Transitionen [* − *Notified* − *Calling*] beschrieben wird. Im Übrigen wird mit der Beziehung

‘*Anruf annehmen*’ **prepares** ‘*Anruf beenden*’

die Tatsache beschrieben, dass ein angenommener Anruf ebenfalls in den Zustand *Calling* führt.

Der mit * markierte Startzustand oben links in Abbildung 6.12 verdient besondere Erwähnung. Dies ist eine abkürzende Schreibweise dafür, dass es keinen eindeutigen Kontrollzustand als Startzustand der Funktionen gibt. Alle in den Funktionen definierten Kontrollzustände kommen als potenzielle Startzustände in Frage. Aus der Perspektive der anderen Funktionen bedeutet dies, dass in einem beliebigen Kontrollzustand der Kontrollfluss übernommen wird. Damit wird eine Unterbrechung oder ein möglicher Abbruch von anderen Funktionen formuliert, das mit den Beziehungstypen interrupt beziehungsweise cancel artikuliert wird. Üblicherweise sind aber nicht alle Kontrollzustände zulässig. Mit der Festlegung entsprechender Vorbedingungen wird die Zustandsmenge begrenzt. Im Falle von «Anruf abweisen» und «Anruf annehmen» werden die Kontrollzustände *Off*, *Locked*, *Blocked*, *Notified* und *Calling* ausgeschlossen.

Der Ausschluss der ersten drei Kontrollzustände ist eine Konsequenz der Beziehung

‘*Einschalten*’ **prepares** ‘*Telefonieren*’.

Darüber hinaus sind mit dieser Spezifikation auch die von den Funktionen selbst verwendeten Kontrollzustände ausgeschlossen. Es ist gewiss nicht sinnvoll, während einer Signalisierung eines Gesprächs ein weiteres Gespräch zu signalisieren. Dass man aber während eines Gesprächs angerufen werden kann, ist nicht unüblich. Man kann, wie bereits oben angedeutet, mit weiteren gesprächsspezifischen Funktionen (zum Beispiel «Makeln») entsprechende Funktionalitäten definieren. Mit dem Ausschluss des Kontrollzustands *Calling* ist dies hier allerdings nicht vorgesehen.

Die Funktion «Anruf annehmen» beschreibt im Wesentlichen die Signalisierung eines ankommenden Gesprächs (`inc_call(number)` auf dem Kanal **sys**) und der anschließenden Bestätigung entweder durch Drücken des „Annahme“ Soft-Keys (siehe nebenstehende Grafik) oder der Abheben-Taste. Die Annahme eines Gesprächs führt stets in den Kontrollzustand *Calling*, aus dem keine Transition zurück in den ursprünglichen Kontrollzustand vorgesehen ist. Jede andere Funktion wird von «Anruf annehmen» abgebrochen. Daher gelten zum Beispiel die Beziehungen

‘*Anruf annehmen*’ **cancels** ‘*PIN eingeben*’ und
‘*Anruf annehmen*’ **cancels** ‘*Navigieren*’.

Auch die Eingabehistorien für «Anruf abweisen» beginnen mit der erwähnten Signalisierungsnachricht, allerdings führt die Abweisung des Gesprächs durch Drücken des „Abweis.“ Soft-Keys oder der Auflegen-Taste wieder zurück in den Startzustand *. Dies ist die abkürzende Schreibweise für die Rückkehr in denselben Kontrollzustand vor der Unterbrechung des Kontrollflusses. Wir schreiben daher zum Beispiel

‘*Anruf abweisen*’ **interrupts** ‘*PIN eingeben*’ und
‘*Anruf abweisen*’ **interrupts** ‘*Navigieren*’,

wenn die unterbrochene Funktion wieder fortgesetzt wird.

Adressen und die Zuordnung zu Rufnummern

Die Adressbuchfunktion, oder kurz «Adressen», mit der Möglichkeit der Verwaltung von Adressen und der damit verbundenen Zuordnung von Rufnummern zu Namen, gehört zum Standardrepertoire eines Mobiltelefons. In dem Zustandsübergangsdiagramm in Abbildung 6.13 greifen wir uns hier wieder drei Funktionen, «Namen anrufen», «Eintrag anlegen» und «Eintrag entfernen», heraus, um die Zusammenhänge zu anderen Funktionen zu veranschaulichen.

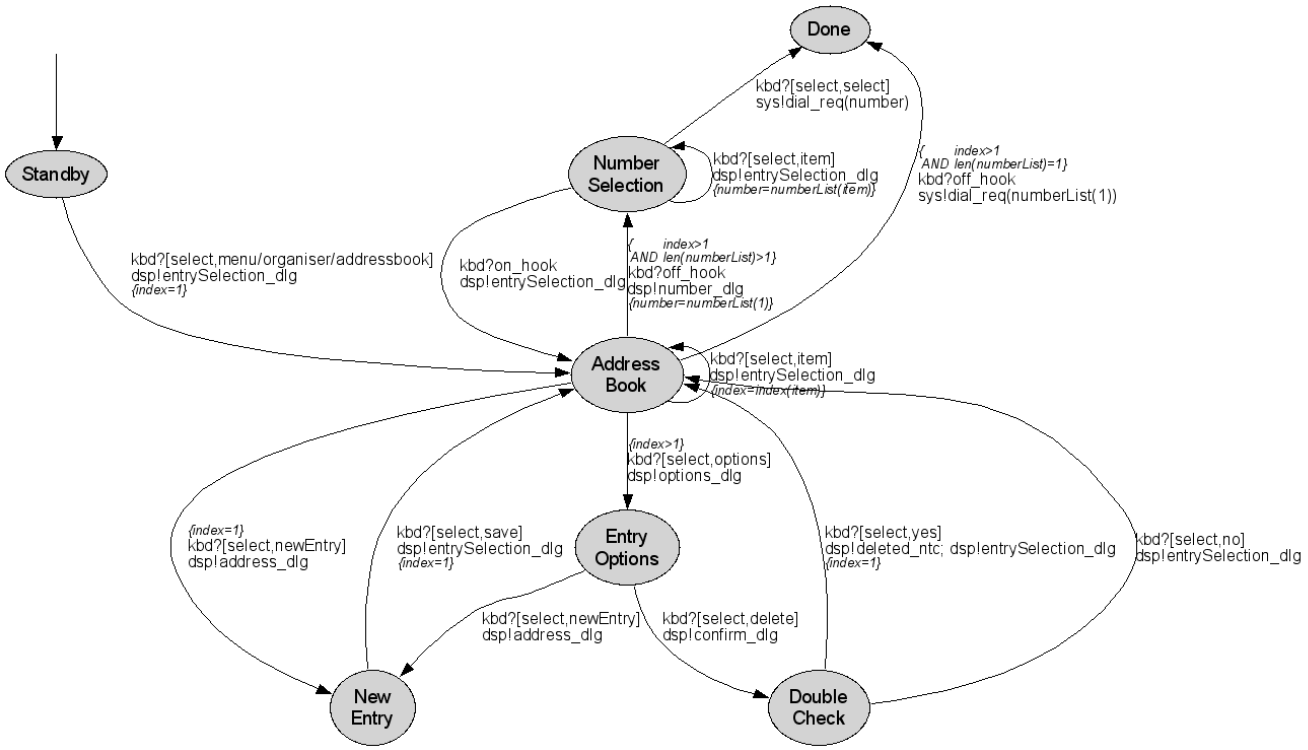


Abbildung 6.13: Adressen mit den Subfunktionen «Eintrag anlegen», «Eintrag entfernen» und «Namen anrufen»

Man erkennt, dass auch hier die Funktionen jeweils ihren Ausgangspunkt in dem Kontrollzustand *Standby* haben. Wie bereits bei der «Navigieren» Funktion erläutert, wird der Zustandsübergang von *Standby* nach *Address Book* durch geeignete Eingaben am Gerät erreicht. Dies wird abkürzend mit `menu/organiser/addressbook` in dem Zustandsübergangsdiagramm beschrieben. Mit den zusätzlichen Eingabe- und Ausgabehistorien wird die Interaktion der «Navigieren» Funktion also ebenfalls ergänzt.

'*Adressen*' **complements** '*Navigieren*'

Allen Subfunktionen gemein ist darüber hinaus die Auswahl eines entsprechenden Adressbucheintrags. Die Steuerung erfolgt über geeignete Eingaben am Gerät und ist hier mit der Eingabenachricht `[select,item]` auf dem Kanal **kbd** repräsentiert. Für diese Spezifikation wird angenommen, dass der erste Eintrag im Adressbuch stets keine Adresse beinhaltet, sondern lediglich die Option zum Anlegen eines neuen Eintrags. Die Selektion des ersten Eintrags wird mit dem Prädikat `index=1` verbunden. Deshalb findet sich im

Zustandsübergangsdiagramm auch die Vorbedingung `{index>1}`, um die Auswahl einer tatsächlichen Adresse zu kennzeichnen.

Die Funktion «Namen anrufen» umfasst zusätzlich die Kontrollzustände *Number Selection* und *Done*. Nach der Auswahl eines Eintrags kann durch Betätigen der Abheben-Taste die Wahl der mit dem Adresseintrag verbundenen Rufnummer ausgelöst werden. Falls dem Namen mehrere Rufnummern zugeordnet sein sollten, dann ist die Vorbedingung `len(numberList)>1` erfüllt und eine der Nummern ist erst auszuwählen. In beiden Fällen wird schließlich über die Ausgabenachricht `dial_req(number)` auf dem Kanal **sys** die Funktion «Nummer wählen» benutzt. Wie bereits bei der Funktion «Nationale Nummer» gesehen, gilt auch hier analog die Beziehung

‘*Namen anrufen*’ **triggers** ‘*Nummer wählen*’.

Die spezifizierten Interaktionen der Funktion «Eintrag anlegen» mit den Kontrollzuständen *Entry Options* und *New Entry* sowie der Funktion «Eintrag entfernen» mit den Kontrollzuständen *Entry Options* und *Double Check* sind einfach nachzuvollziehen. Wir gehen deshalb hier nicht im Detail auf die Funktionen ein. Allerdings beeinflussen sie durchaus andere Funktionen in ihrem Ausgabeverhalten. Existiert etwa ein Eintrag im Adressbuch, so ändern sich im Falle eines Anrufs einige Ausgabenachrichten.

Die gültigen Eingaben der beeinflussten Funktion sind unverändert, jedoch hängen deren beobachtbaren Ausgabevarianten von der anderen Funktion ab. In dem in Abbildung 6.12 dargestellten Zustandsübergangsdiagramm sind die Ausgabenachrichten der Funktionen «Anruf annehmen», «Anruf abweisen» und «Nummer wählen» abhängig von der Vorbedingung `existAddr(number)`. Existiert eine Zuordnung einer Nummer zu einem Adressbucheintrag, so wird statt der Rufnummer der entsprechende Name angezeigt. Die nebenstehende Grafik zeigt zum Beispiel die geänderte Displayausgabe bei der Rufannahme. Aufgrund der Beziehungen

‘*Adressen*’ **refines** ‘*Nummer wählen*’,
‘*Adressen*’ **refines** ‘*Anruf annehmen*’ und
‘*Adressen*’ **refines** ‘*Anruf abweisen*’.

sind in dem Zustandsübergangsdiagramm der «Telefonieren» Funktion die Transitionen [* − *Notified*] und [*Check Number* − *Dialling*] jeweils zusammen mit den Vorbedingungen `existAddr(number)` eingeführt worden. Das Prädikat `existAddr` stellt im Grunde ebenfalls eine Subfunktion von «Adressen» dar. Da diese Funktion aber nur über interne Kanäle genutzt wird und insbesondere keine beobachtbaren Interaktionen nach sich zieht, wurde hier auf eine detaillierte Spezifikation verzichtet.

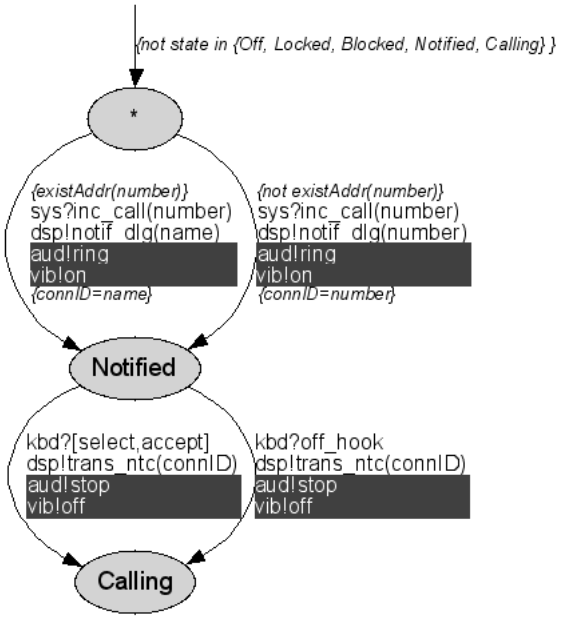


Abbildung 6.14: Anruf mit mehrfacher Signalisierung annehmen

Mehrfache Signalisierung eines Anrufs

In der in Abbildung 6.12 spezifizierten Form wird bei einem eingehenden Anruf nur auf dem Display eine entsprechende Notiz angezeigt werden (`notif_dlg(...)` auf dem Kanal **dsp**). Auf die Angabe weiterer Ausgaben, wie zum Beispiel das übliche Abspielen eines Klingeltons, wurde dort der Übersichtlichkeit halber verzichtet.

Im Hinblick auf die Unterstützung verschiedener Modellvarianten können nun bestimmte Typen einer Mobiltelefonserie etwa auch Anrufe durch das Vibrieren des Geräts anzeigen. Der Ausschnitt eines Zustandsübergangsdiagramms in Abbildung 6.14 zeigt nun solch eine Funktion. In den dunkel hinterlegten Bereichen sind entsprechende Ausgaben auf den zusätzlichen Kanälen markiert. Kombinierte man die beiden Funktionen «Anruf annehmen» und «Anruf mit mehrfacher Signalisierung annehmen», so kommen zu gegebenen Eingabehistorien nur noch die Ausgabehistorien der zuletzt genannten Funktion zum Tragen. Es gilt die Beziehung

‘*Anruf mit mehrfacher Signalisierung annehmen*’ **replaces** ‘*Anruf annehmen*’.

Nebenbei bemerkt würden in der Kombination der Funktionen nach wie vor die Ausgaben an der Schnittstelle der Funktion «Anruf annehmen» gleich bleiben. Denn projiziert man die Ausgaben der Superfunktion «Telefonieren» auf die Schnittstelle von «Anruf annehmen», ist keine Verhaltensänderung zu beobachten. Für alle $x \in \mathsf{dom}(\mathit{Telefonieren})$ gilt nämlich

$$\begin{aligned}&(\mathit{Telefonieren}.x) \mid O_{\mathit{Anruf\ annehmen}} \\ &\quad = \mathit{Telefonieren} \dagger (I_{\mathit{Anruf\ annehmen}} \blacktriangleright O_{\mathit{Anruf\ annehmen}}).(x \mid I_{\mathit{Anruf\ annehmen}})\end{aligned}$$

Anzeigen des Betriebszustands

Auch anhand der vergleichsweise einfachen Funktion «Betriebszustand anzeigen» lässt sich eine replace Beziehung illustrieren. Diese Funktion wurde im vorhergehenden Abschnitt bereits im Zusammenhang mit dem Begriff „Bereitschaftszustand des Geräts“ beziehungsweise mit dem Kontrollzustand *Standby* angesprochen.

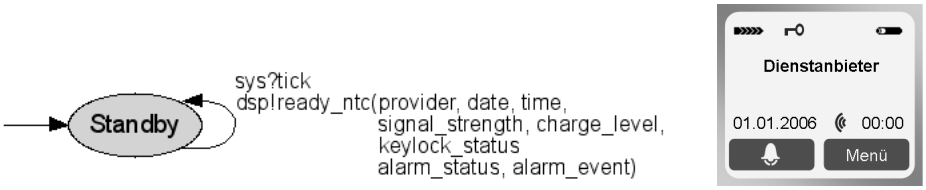


Abbildung 6.15: Betriebszustand anzeigen

Die Funktion «Betriebszustand anzeigen» kommt nicht auf Initiative des Benutzers zur Anwendung. Vielmehr wird im Leerlaufbetrieb aufgrund einer Systemnachricht (`tick` auf dem Kanal **sys**) in regelmäßigen Abständen eine aktuelle Information angezeigt. In dem in Abbildung 6.15 dargestellten Zustandsübergangsdiagramm wird dies durch die Nachricht `ready_ntc(...)` ausgedrückt.

In der einfachsten Form werden lediglich der Name des aktuellen Dienstanbieters, Datum und Uhrzeit, die Signalstärke sowie der Ladezustand des Akkus auf dem Display angezeigt. Die hier dargestellte Spezifikation bringt allerdings bereits die umfassendere Variante einer Standby-Anzeige mit sich. Die einzelnen Parameter der Ausgabenachricht können leicht den Symbolen auf der Display-Darstellung zugeordnet werden. Die letzten drei Größen `keylock_status`, `alarm_status` und `alarm_event` sind boolesche Werte, die noch im Zusammenhang mit den Funktionen «Tastensperre» und «Wecker» näher betrachtet werden.

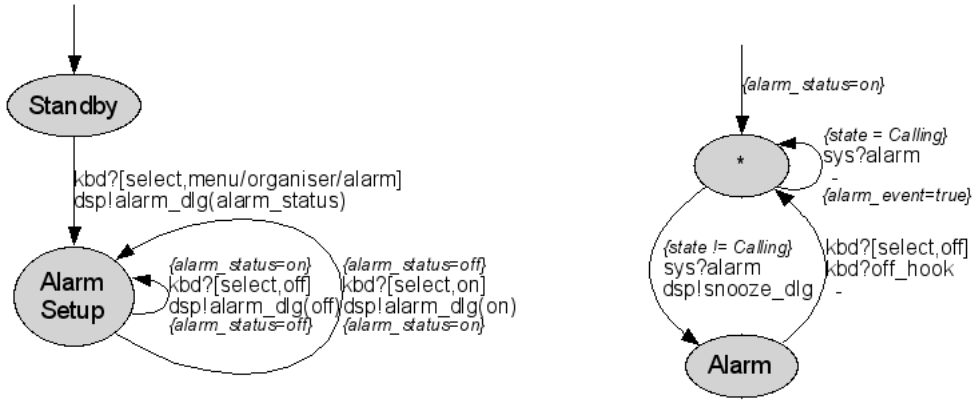
Da in der Kombination einer Funktion mit anspruchsloser Anzeige (etwa «Einfache Anzeige») und der hier aufgeführten Funktion mit umfangreicherer Anzeige die Ausgaben

ersetzt werden, gilt die Beziehung

'*Betriebszustand anzeigen*' **replaces** '*Einfache Anzeige*'

Die Weckfunktion

Eine weitere typische Funktionalität eines Mobiltelefons ist die Alarm oder «Wecker» Funktion. Obwohl sie im Vergleich zu den bisher gezeigten Funktionen verhältnismäßig einfach ist, können trotzdem eine Reihe von speziellen Funktionsbeziehungen illustriert werden.



(a) Wecker aktivieren (b) Wecker

Abbildung 6.16: Alarm „Bundle“

Wird eine Weckzeit eingestellt und aktiviert, so erfolgt zu dem eingestellten Zeitpunkt eine entsprechende Hinweismeldung und zwar prinzipiell in einem beliebigen Zustand des Geräts. In Abbildung 6.16 sind die Zustandsübergangsdiagramme der Funktionen «Wecker aktivieren» und «Wecker» dargestellt. Die gemeinsame Betrachtung der Funktionsspezifikationen soll im Übrigen verdeutlichen, dass das „Bundling“ der Funktionen losgelöst von dem Superfunktionsbegriff ist. Beide Funktionen besitzen unterschiedliche Superfunktionen, wobei die kleinste umfassende Superfunktion «Mobiltelefon» selbst ist. Der Zugang zu beiden Funktionen in Kombination wird also über die Funktion «Mobiltelefon» definiert.

Analog zu den oben beschriebenen Erweiterungen der «Navigieren» Funktion wird auch mit der Funktion «Wecker aktivieren» der Zustandsübergang von *Standby* nach *Alarm Setup* durch geeignete Eingaben, hier `menu/organiser/alarm`, beschrieben. Mit diesen

zusätzlichen Eingabe- und Ausgabehistorien wird die Interaktion der «Navigieren» Funktion ebenso ergänzt.

‘*Wecker aktivieren*’ **complements** ‘*Navigieren*’

Im Zustand *Alarm Setup* kann der Wecker beliebig oft ein- und ausgeschaltet werden (`[select,on]`, `[select,off]` auf dem Kanal **kbd**). Genau genommen wird mit diesem Zustandsübergangsdiagramm nicht nur die Funktion «Wecker aktivieren», sondern zugleich auch die Deaktivierung beschrieben. Nur die Eingabehistorien, welche als letzte Nachricht `[select,on]` beinhalten, sind Elemente der Domäne von «Wecker aktivieren». Das Ergebnis ist eine Änderung des Datenzustands, indem die Variable `alarm_status` auf `on` gesetzt wird. Die Funktion «Wecker aktivieren» beeinflusst damit auch die Ausgabe der Funktion «Betriebszustand anzeigen» (siehe Abbildung 6.15). Es gilt

‘*Wecker aktivieren*’ **refines** ‘*Betriebszustand anzeigen*’,

da über den Parameter `alarm_status` nur bestimmte Ausgabehistorien zugelassen werden. Der Wert der `alarm_status` Variable ist zugleich bestimmend für die «Wecker» Funktion. Wie man in Abbildung 6.16(b) erkennen kann, ist sie nämlich nur definiert, wenn das Prädikat `alarm_status=on` wahr ist. Da die Funktion «Wecker aktivieren» dafür verantwortlich ist, wird dies mit der Beziehung

‘*Wecker aktivieren*’ **enables** ‘*Wecker*’

ausdrückt. Es gibt eine weitere, hier nicht näher spezifizierte Subfunktion, welche unter der Bedingung `alarm_status=on` das Signal `alarm` auf dem Kanal **sys** erzeugt.

Ähnlich wie bei der «Telefonieren» Funktion gibt es für die «Wecker» Funktion keinen eindeutigen Kontrollzustand als Startzustand. Hier wird wieder die abkürzende Schreibweise eines mit * markierten Startzustands verwendet. Im Gegensatz zur «Telefonieren» Funktion sind hier tatsächlich alle in den Funktionen definierten Kontrollzustände als potenzielle Startzustände zulässig. Dies gilt insbesondere auch für den Zustand *Off* – ein Alarm ist somit auch für ein ausgeschaltetes Gerät spezifiziert.

Auch führen alle Eingabehistorien der «Wecker» Funktion wieder zurück in den Startzustand *. Daher gelten zum Beispiel die Beziehungen

‘*Wecker*’ **interrupts** ‘*Telefonieren*’,
‘*Wecker*’ **interrupts** ‘*Navigieren*’ und
‘*Wecker*’ **interrupts** ‘*Einstellen*’.

Das Ausgabeverhalten ist abhängig vom Startzustand. Ein Telefongespräch, das durch den Kontrollzustand *Calling* repräsentiert wird, soll nicht durch einen Alarmhinweis gestört werden. In jedem anderen Fall wird die Funktion sichtbar unterbrochen, unter anderem mit einem Bestätigungsdialog (siehe nebenstehende Grafik), der durch Drücken des „Aus“ Soft-Keys oder der Auflegen-Taste beendet wird (`[select,off]` oder `off_hook` auf dem Kanal **kbd**). In dem Zustandsübergangsdiagramm der «Wecker» Funktion in Abbildung 6.16(b) ist der Einfachheit halber nur das Display mit der Kanalbezeichnung **dsp** als Ausgabemedium spezifiziert. In der Realität wird meist eine audiovisuelle Signalisierung ausgelöst.

Ereignet sich nun während eines Telefonats eine Alarmsignalisierung (`state = Calling`), so ist die Unterbrechung für den Nutzer nicht unmittelbar zu beobachten. Jedoch wird der Datenzustand geändert und der Variablen `alarm_event` der Wert `true` zugewiesen. Wie bereits bei der Funktion «Wecker aktivieren» gilt sinngemäß auch hier die Beeinflussung des Ausgabeverhaltens der Funktion «Betriebszustand anzeigen». In der nebenstehenden Grafik ist die entsprechende Ausgabenachricht `ready_ntc(..., alarm_event)` mit der Anzeige eines Glockensymbols unten links angedeutet. Es gilt

‘*Wecker*’ **refines** ‘*Betriebszustand anzeigen*’.

Die Änderung des Schnittstellenverhaltens wird demnach von der «Telefonieren» Funktion veranlasst. Die «Wecker» Funktion wird also auch beeinflusst. Um der nicht sichtbaren Unterbrechung genüge zu tun, ist im Zusammenhang mit der «Telefonieren» Funktion die Domäne von «Wecker» entsprechend zu erweitern und zugleich sind die Eingabeoptionen abhängig vom Kontrollzustand zu spezifizieren. Dann gilt

‘*Telefonieren*’ **directs** ‘*Wecker*’.

Die Wahlwiederholung

Die einfachste Variante der Wahlwiederholung deckt die Funktion «Letzte Nummer wiederholen» ab. Durch zweimaliges Drücken der Abheben-Taste wird die zuletzt gewählte Nummer angerufen. In dem Zustandsübergangsdiagramm in Abbildung 6.17 wird dies an den Transitionen [*Standby* − *Redial*] und [*Redial* − *Done*] jeweils mit der Eingabenachricht `off_hook` auf dem Kanal **kbd** sichtbar.

Die Ausgabenachricht der Transition [*Redial* − *Done*] weist, wie bereits bei den Funktionen «Nationale Nummer» oder «Namen anrufen» gesehen, auf die Nutzung der Funktion

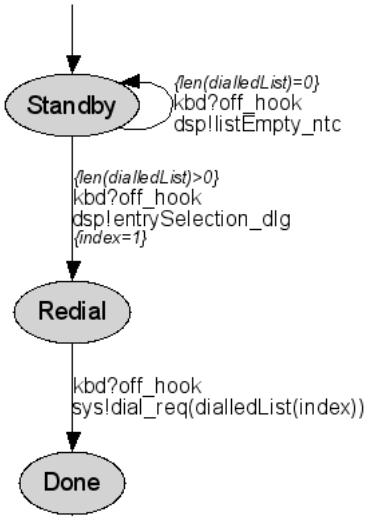


Abbildung 6.17: Letzte Nummer wiederholen

«Nummer wählen» hin. Auch hier gilt die Beziehung

'*Letzte Nummer wiederholen*' **triggers** '*Nummer wählen*'.

Isoliert betrachtet kann die oben erwähnte Eingabesequenz `<off_hook, off_hook>` auch zu einem anderen Ausgabeverhalten führen, etwa zu der Nachrichtensequenz `<listEmpty_ntc, listEmpty_ntc>` auf dem Kanal **dsp**. Die Präzisierung des Verhaltens wird in Kombination mit anderen Funktionen erreicht, welche für die Wirksamkeit der angegebenen Vorbedingungen verantwortlich sind. Wenn wir uns die Funktion «Nummer wählen» in Erinnerung rufen (siehe Abbildung 6.12), wird dort bereits als Nachbedingung der Transition [*Check Number − Dialling*] die Rufnummernliste mit `dialledList(1)=number` aktualisiert. Es gilt die Beziehung

'*Nummer wählen*' **refines** '*Letzte Nummer wiederholen*',

da für ein und dieselbe Eingabehistorie die Menge der Ausgabehistorien reduziert wird.

Die Wiederholung der Wahl einer Rufnummer ist nicht grundsätzlich auf die zuletzt gewählte Nummer eingeschränkt. Typischerweise wird eine größere Menge an gewählten Nummern in einer zeitlich sortierten Liste gespeichert, wobei die zuletzt verwendete Nummer an erster Stelle steht (`index=1`). Das Zustandsübergangsdiagramm in Abbildung 6.18

zeigt nun unter anderem zusätzlich die Funktion «Beliebige Nummer wiederholen». Diese ist zunächst gekennzeichnet durch weitere Transitionen, die aus dem Kontrollzustand *Redial* führen, um die Auswahl aus einer Rufnummernliste abzubilden.

Darüber hinaus bietet sie eine weitere Zugangsmöglichkeit über den bereits bekannten Mechanismus der Menüauswahl (`[select,menu/records]` auf dem Kanal **kbd**). Auch hier wird dies mit der Beziehung

'*Beliebige Nummer wiederholen*' **complements** '*Navigieren*'

charakterisiert.

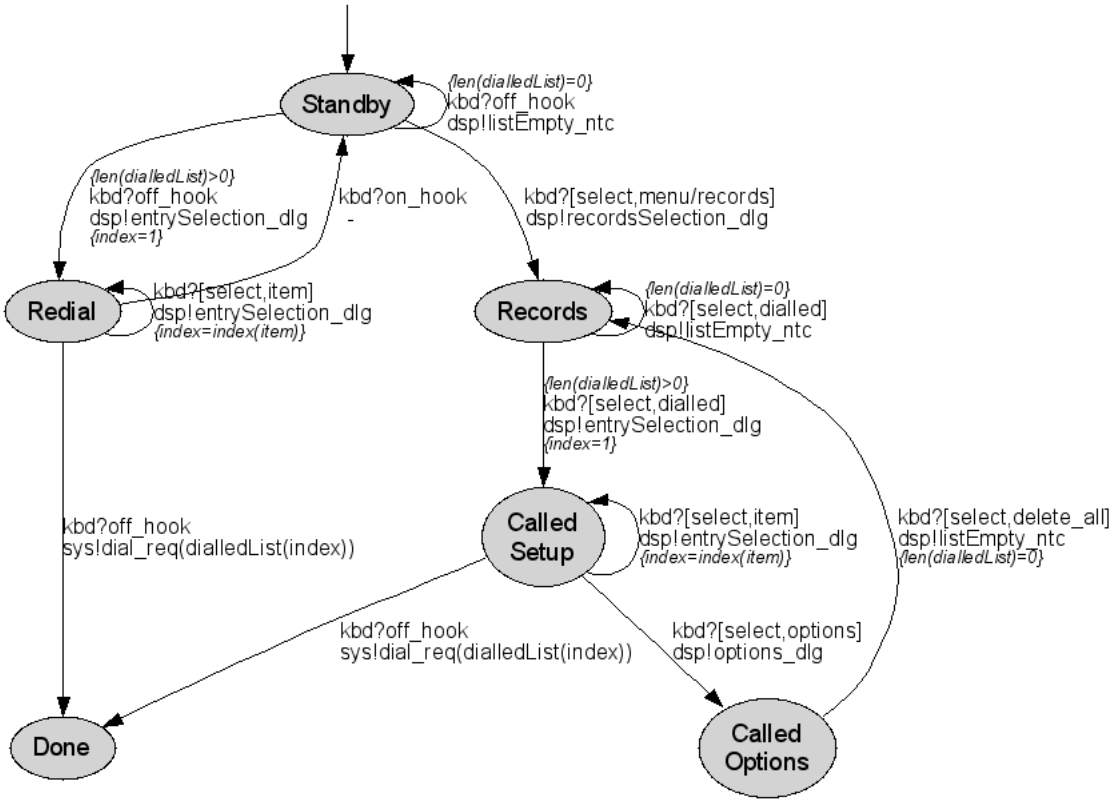


Abbildung 6.18: Wahl wiederholen und gewählte Nummern löschen (Ruflisten „Bundle“)

Im Kontrollzustand *Records* stellt sich eine ähnliche Situation dar, wie sie bereits im Zusammenhang mit der Funktion «Letzte Nummer wiederholen» und der Reduzierung der

Ausgabehistorien besprochen wurde. In Abhängigkeit von Vorbedingungen, für deren Erfüllung kombinierte Funktionen ausschlaggebend sind, werden bestimmte Eingabehistorien und damit entsprechende Ausgabehistorien zugelassen. Hier ist es so, dass die Zuordnung der Eingabehistorien zu Ausgabehistorien eindeutig ist. Mit der Beziehung

'*Nummer wählen*' **directs** '*Beliebige Nummer wiederholen*'

wird dies zum Ausdruck gebracht. Nach der Auswahl einer Rufnummer aus der Liste im Kontrollzustand *Called Setup* wird durch Drücken der Abheben-Taste ebenso die Funktion «Nummer wählen» genutzt. Nachdem beide Subfunktionen der «Wahlwiederholung» in derselben Beziehung stehen, schreiben wir auch

'*Wahlwiederholung*' **triggers** '*Nummer wählen*'.

Der Vollständigkeit halber sei erwähnt, dass die kombinierte Spezifikation des Ruflisten Bundles in Abbildung 6.18 auch die «Einstellen»-Subfunktion «Gewählte Nummern löschen» umfasst. Mit den Transitionen [*Standby* − *Records* − *Called Setup* − *Called Options*] ergänzt auch sie die «Navigieren» Funktion.

'*Gewählte Nummer löschen*' **complements** '*Navigieren*'

Da ebenso wie die Funktion «Nummer wählen» auch die Funktion «Gewählte Nummern löschen» die möglichen Eingabehistorien und Ausgabehistorien beeinflusst, gelten auch hier die Beziehungen

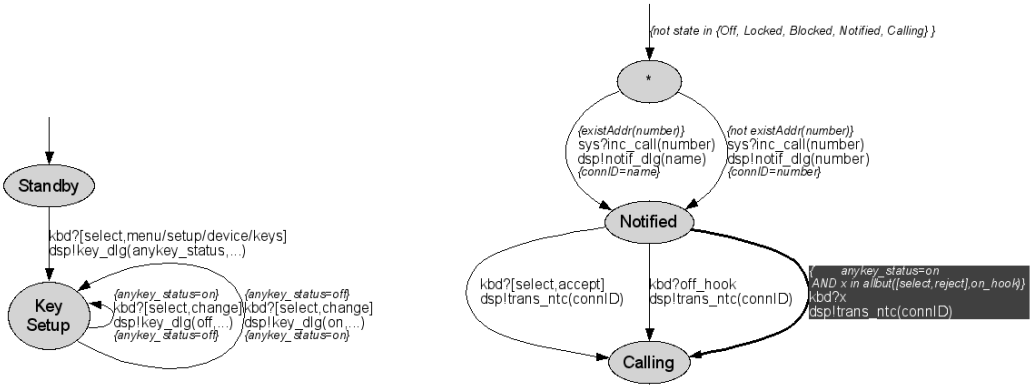
'*Gewählte Nummern löschen*' **refines** '*Letzte Nummer wiederholen*' und
'*Gewählte Nummern löschen*' **directs** '*Beliebige Nummer wiederholen*'.

Rufannahme mit beliebigen Tasten und gesperrten Tasten

Zum Abschluss untersuchen wir noch ein Beispiel für die Kombination von drei interagierenden Funktionen. Damit soll demonstriert werden, dass die Identifizierung bilateraler Beziehungen eine grundlegende und wichtige Maßnahme im Zuge eines vorbildlichen Requirements Engineerings ist. Für die Bewertung potenzieller Feature Interactions komplexerer Zusammenhänge ist dies jedoch nicht immer ausreichend. Das Vorhandensein eines entsprechenden Beziehungskalküls ist dafür unumgänglich.

In Abbildung 6.19(a) ist das Zustandsübergangsdiagramm für die Aktivierung und Deaktivierung der erweiterten Rufannahme, kurz «Jede Taste», dargestellt. Wie bereits mehrfach vorher gezeigt, gilt auch hier entsprechend die Beziehung

'*Jede Taste*' **complements** '*Navigieren*'.



(a) Jede Taste einstellen

(b) Jede Taste zur Anrufannahme

Abbildung 6.19: Jede Taste

Der Leser rufe sich an der Stelle die Funktion «Wecker aktivieren» in Erinnerung. Die dort beschriebenen Transitionen auf Seite 164 wirken analog. Wir gehen daher nicht weiter auf die einzelnen Transitionen ein. Entscheidend ist das Ergebnis von «Jede Taste» mit der Änderung des Datenzustands, wobei die Variable `anykey_status` entweder auf `on` oder auf `off` gesetzt wird.

Mit «Jede Taste» wird auch zugleich die Menge der Eingabehistorien der «Anruf annehmen» Funktion vergrößert. Durch die zusätzlich definierte Transition (siehe Abbildung 6.19(b) dunkel hinterlegt im Vergleich zu Abbildung 6.12) sind zur Annahme eines Anrufs sämtliche Tasten, mit Ausnahme derjenigen zur Rufabweisung, zulässige Eingaben. Wir formulieren dies daher mit der Beziehung

‘*Jede Taste*’ **extends** ‘*Anruf annehmen*’.

Die andere Funktion, die wir untersuchen möchten, ist die «Tastensperre». Das Zustandsübergangsdiagramm in Abbildung 6.20 zeigt diese Funktion mit den Subfunktionen «Tasten sperren» und «Tasten entsperren». Sie ändert ähnlich wie die «Wecker aktivieren» Funktion den Datenzustand mit der Zuweisung der Werte `on` beziehungsweise `off` zu der Variablen `keylock_status`. Der Vollständigkeit halber sei erwähnt, dass hiermit ebenfalls das Ausgabeverhalten der Funktion «Betriebszustand anzeigen» beeinflusst wird. In der Grafik in Abbildung 6.15 ist die entsprechende Ausgabenachricht für `keylock_status=on` mit der Anzeige eines Schlüsselsymbols oben links angedeutet. Es gilt

‘*Tastensperre*’ **refines** ‘*Betriebszustand anzeigen*’.

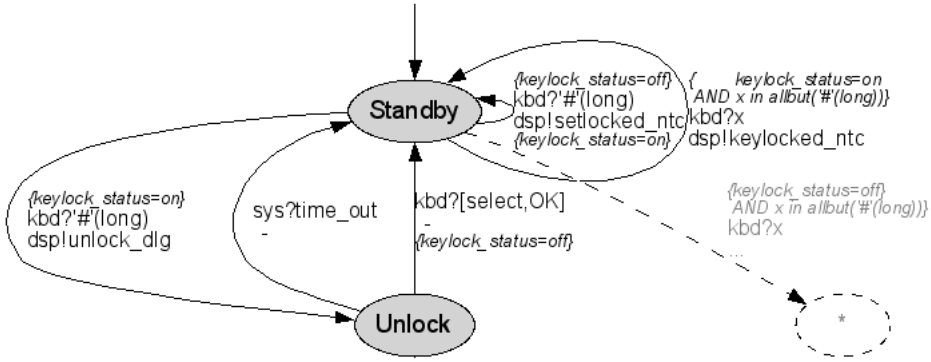


Abbildung 6.20: Tastensperre mit den Subfunktionen «Tasten sperren» und «Tasten entsperren»

Das Verhalten der Subfunktion «Tasten entsperren» ist durch die drei Transitionen zwischen den Kontrollzuständen *Standby* und *Unlock* beschrieben. Für die definierte Anwendung dieser Funktion ist einerseits die in der Domäne festgelegte korrekte Bedienung erforderlich: jede zulässige Eingabehistorie weist als erste Nachricht `'#'(long)` auf dem Kanal **kbd** auf (langes Drücken der #-Taste). Andererseits muss auch der Zugang zur der Funktion gegeben sein, der durch die Vorbedingung `keylock_status=on` festgelegt ist. Dieser Zugang wird nun über die Funktion «Tasten sperren» ermöglicht und wir schreiben daher

'*Tasten sperren*' **enables** '*Tasten entsperren*'.

Auf der anderen Seite wird aber der Zugang von der Funktion selbst verhindert und bedeutet konsequenterweise

'*Tasten entsperren*' **disables** '*Tasten entsperren*'.

Wenden wir uns nun der Funktion «Tasten sperren» zu. In Abbildung 6.20 wird das Verhalten im Wesentlichen zunächst mit den beiden Transitionen beschrieben, die jeweils als Ausgangszustand und Folgezustand den *Standby* Kontrollzustand besitzen. Ist die Sperrung der Tasten noch nicht aktiviert (`keylock_status=off`), kann die Sperre ebenfalls durch langes Drücken der #-Taste veranlasst werden.

Bei aktivierter Sperrung der Tasten führt, mit Ausnahme der lang gedrückten #-Taste fürs Entsperren, jede Tastatureingabe lediglich zu einem Hinweis, dass die Tastatur gesperrt ist (`keylocked_ntc` auf dem Kanal **dsp**). Für alle anderen Funktionen, die aus dem *Standby* Kontrollzustand heraus Eingabenachrichten der Tastatur erfordern, wird damit der Zugang verhindert. Es gilt beispielsweise

‘*Tasten sperren*’ **disables** ‘*Navigieren*’,
‘*Tasten sperren*’ **disables** ‘*Nationale Nummer*’,
‘*Tasten sperren*’ **disables** ‘*Adressen*’ und
‘*Tasten sperren*’ **disables** ‘*Wahlwiederholung*’.

Dies ist mit der gestrichelt markierten Transition in Abbildung 6.20 angedeutet: der Zugang zu diesen Funktionen ist nur dann gewahrt, wenn die Tastensperre nicht gesetzt ist. Im Übrigen treffen auch die dualen enable Beziehungen für die Funktion «Tasten entsperren» zu.

Die Dreierkombination

Soweit zu den bilateralen Beziehungen der Funktionen. Wie verhält sich aber nun die Funktion «Anruf annehmen» in der Kombination mit «Jede Taste» und «Tasten sperren»? Gemäß der oben beschriebenen Restriktion des Zugangs ist selbstverständlich auch die Funktion «Jede Taste» betroffen. Es gilt ebenso

‘*Tasten sperren*’ **disables** ‘*Jede Taste*’

und die erweiterte Rufannahme kann im Falle gesperrter Tasten nicht eingestellt werden. Die Tastensperre hat dagegen laut Spezifikation keinen Einfluss auf die Rufannahme, denn diese Eingabehistorien weisen als erste Nachricht `inc_call(number)` auf dem Kanal **sys** auf, das heißt

independent(*Anruf annehmen*, *Tasten sperren*).

Wenn wir aber annehmen, dass die Tasten nicht gesperrt sind und die Rufannahme mit «Jede Taste» erweitert worden ist, welche Auswirkung hat dann die Sperrung der Tasten?

Eine mögliche Verhaltensvariante in dem Fall ist, dass ein Ruf dann nicht mehr mit jeder Taste angenommen werden kann. Die Tastensperrung neutralisiert gleichsam die Erweiterung der Rufannahme. Genau dieses Verhalten ist in dem Zustandsübergangsdiagramm in Abbildung 6.21 spezifiziert. Dort ist das kombinierte Verhalten der Funktion «Anruf annehmen» mit den beiden Funktionen berücksichtigt. Die Konjunktion als Vorbedingung der Transition [*Notified* − *Calling*] ist dort um den Term `keylock_status=off` ergänzt worden (in der Abbildung dunkel hinterlegt). Ist also «Anruf annehmen» doch

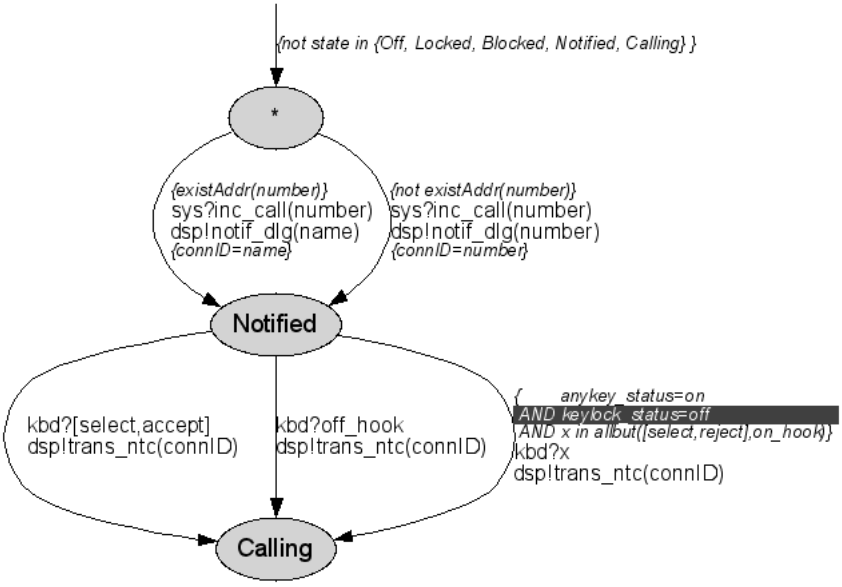


Abbildung 6.21: Anrufannahme in Kombination mit «Jede Taste» und «Tastensperre»

nicht unabhängig von «Tasten sperren»?

Die Antwort ist klar: die Unabhängigkeit dieser Funktionen ist nach wie vor gegeben. Wenn man allerdings die erweiterte Rufannahme zugrunde legt, so ist die Beeinflussung offensichtlich. Es ist allerdings nicht notwendig, dafür eine weitere Beziehung zu formulieren. Die oben genannte Beziehung, '*Tasten sperren*' **disables** '*Jede Taste*', ist ausreichend. Sie gilt also nicht nur für das Einstellen von «Jede Taste», sondern auch für den operativen Teil der Erweiterung der Rufannahme.

Im Hinblick auf die Entwicklung eines Kalküls von Beziehungen ist etwa folgende Regel denkbar. „Wenn eine Funktion h existiert, die den Zugang zu f beschränkt, so kann die Funktion f eine Funktion g nicht erweitern“:

$$\exists\, h : \mathbf{disable}(h, f) \Rightarrow \neg\mathbf{extend}(f, g).$$

6.5 Unabhängige Funktionen und implizite Beziehungen

Bevor wir abschließend die analysierten Funktionsbeziehungen der Fallstudie in ihrer Gesamtheit diskutieren, sollten vorher noch zwei Aspekte erwähnt werden. Einerseits findet sich in dieser Fallstudie bis auf eine Ausnahme keine Unabhängigkeitsbeziehung zwischen zwei Funktionen. Wir werden dies gleich anschließend begründen. Andererseits ist auch nicht offensichtlich, welcher Art von Zusammenhang zwischen Subfunktionen gilt, wenn es eine Beziehung zwischen deren Superfunktionen geben sollte. Wir beschreiben an dieser Stelle nur qualitativ die Konsequenzen implizit gegebener Beziehungen. Letztendlich gehören solche Beziehungen ebenfalls in den Kontext eines Beziehungskalküls gestellt, das jedoch außerhalb des Rahmens dieser Arbeit liegt.

Unabhängigkeit von Funktionen

In den vorhergehenden Abschnitten konnten die Zusammenhänge aller für die Fallstudie ausgewählten Funktionen mit dem in Kapiteln 4 und 5 vorgestellten Satz von Beziehungstypen beschrieben werden. Allerdings wird dem Leser aufgefallen sein, dass ein Beziehungstyp dabei nur einmal explizit verwendet wurde, und zwar die Unabhängigkeit einer Funktion zu einer anderen (independent).

Eine Interpretation bestünde nun darin, dass alle nicht spezifizierten Zusammenhänge zwischen Funktionen grundsätzlich als independent angenommen werden. Dies erscheint allerdings sehr zweifelhaft, solange kein geeigneter Kalkül existiert, mit dessen Hilfe die Zusammenhänge der Funktionen geeignet verifiziert werden können.

Die empfohlene Annahme ist deshalb, dass für nicht spezifizierte Funktionszusammenhänge diese Beziehungen (noch) nicht definiert sind. Der independent-Beziehungstyp kommt nur dann zum Einsatz, wenn explizit die Unabhängigkeit einer Funktion von einer anderen Funktion gefordert wird. Dies wird insbesondere dann interessant, wenn es einen Beziehungskalkül gibt und damit unvereinbare Beziehungen zwischen Funktionen bereits bei der Anforderungsanalyse identifiziert werden können. Beispielsweise sollte es im Rahmen der Funktionen der Fallstudie nicht zulässig sein, wenn die zunächst nahe liegende Unabhängigkeit zwischen den Funktionen «PIN eingeben» und «Telefonieren» gefordert sein sollte,

independent(*PIN eingeben*, *Telefonieren*)

aber sich zugleich deren Subfunktionen sehr wohl beeinflussen, zum Beispiel

‘*Anruf annehmen*’ **cancels** ‘*Korrekte PIN eingeben*’.

Implizite Beziehungen

Wenn eine Beziehung zwischen zwei Funktionen festlegt wird, gehen wir davon aus, dass damit auch Beziehungen desselben Typs jeweils für alle entsprechenden Subfunktionen gelten. Gilt beispielsweise die Beziehung

'*Einschalten*' **prepares** '*Navigieren*'

so ist dies gleichbedeutend mit

'*Einschalten mit Einschaltsicherung*' **prepares** '*Zurücksetzen*',
'*Einschalten mit Einschaltsicherung*' **prepares** '*Thema auswählen*',
'*Einschalten mit Zugriffsschutz*' **prepares** '*Zurücksetzen*' und
'*Einschalten mit Zugriffsschutz*' **prepares** '*Thema auswählen*'.

Dies leuchtet sofort ein, denn mit der «Einschalten» Funktion wird ein neuer Betriebsmodus erreicht, in dem auch dann erst die entsprechenden (Sub-) Funktionen des Navigierens zur Verfügung stehen.

Ein weiteres Beispiel für implizite Beziehungen können wir auch beim Zusammenhang zwischen der «Navigieren» Funktion und den „Plug-In" Funktionen für das Einstellen anderer Funktionen feststellen. Im Vergleich zu dem obigen Beispiel mit prepare sollte hier die Semantik allerdings eine andere sein. Wenn etwa

'*Einstellen*' **complements** '*Navigieren*'

gilt, so kann dies hier nicht bedeuten, dass alle Subfunktionen von «Einstellen» die Interaktionen aller Subfunktionen von «Navigieren» ergänzen. Statt des Allquantors sollte hier besser der Existenzquantor zum Tragen kommen. Die Bedeutung liegt also vielmehr darin, dass es mindestens eine Subfunktion von «Einstellen» gibt, welche eine Interaktion von mindestens einer Subfunktion von «Navigieren» ergänzt.

Wie man sieht, ist es nicht unbedingt von Vorteil, für alle Beziehungstypen dieselbe Ableitungssemantik vorzuschreiben. Hier gilt es, die Semantik einer Beziehung zwischen Superfunktionen, die abgeleiteten Beziehungen für deren Subfunktionen sorgfältig zu analysieren und zu definieren. Auch im umgekehrten Fall, wenn Subfunktionen in einer Beziehung stehen, ist die abgeleitete Beziehung der jeweiligen Superfunktionen abhängig vom Beziehungstyp.

6.6 Die Nutzungsarchitektur der Mobiltelefonfunktionen

Man kann sich leicht vorstellen, dass selbst bei einer überschaubaren Menge von Funktionen, wie sie etwa bei dieser Fallstudie angenommen wurde, die Nutzungsarchitektur

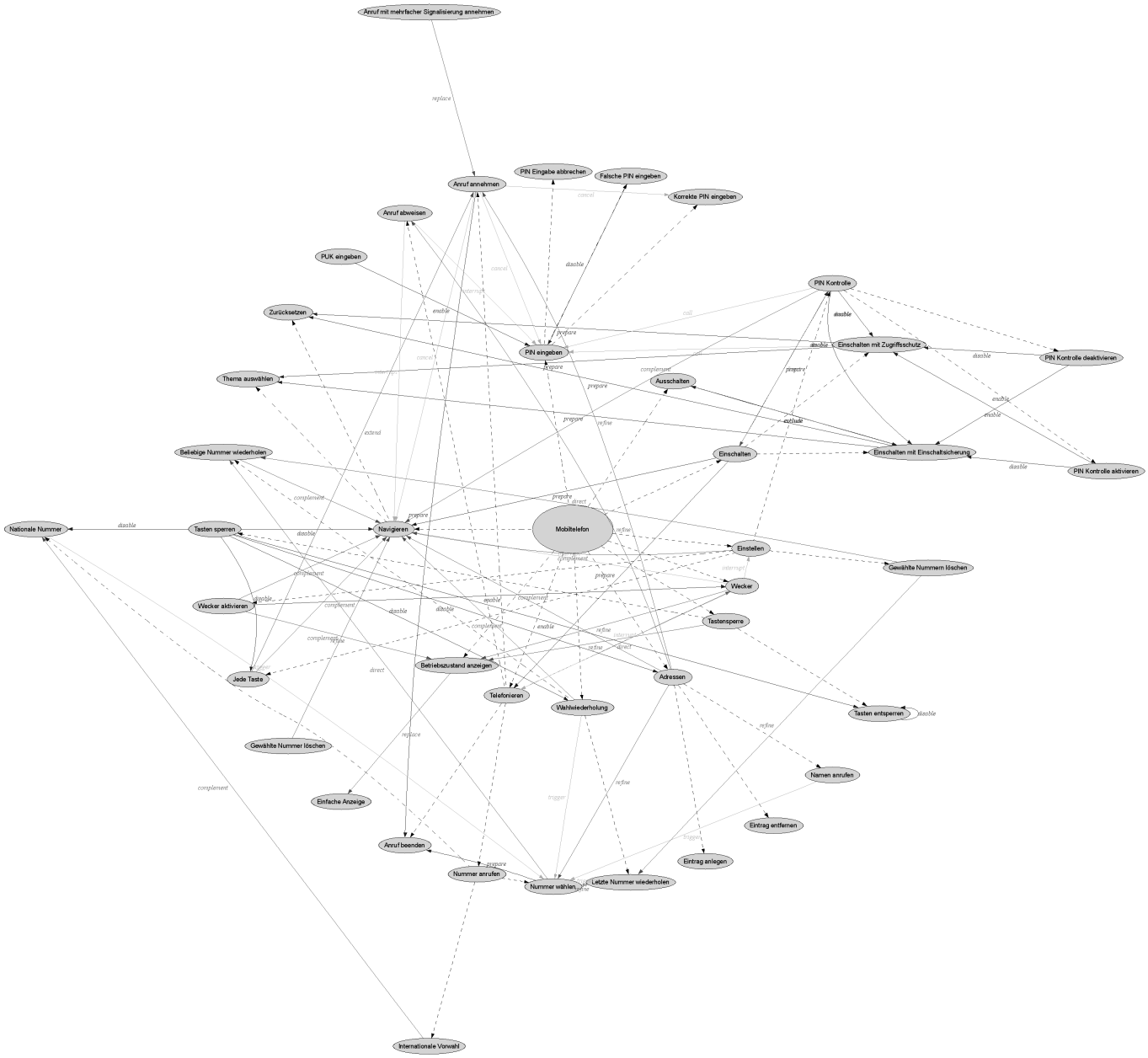


Abbildung 6.22: Nutzungsarchitektur der ausgewählten Nutzungsfunktionen eines Mobiltelefons

bereits eine hohe Komplexität aufweisen kann. Die Übersichtlichkeit kann insbesondere bei graphischen Darstellungen schnell verloren gehen. In Abbildung 6.22 ist dies auch unschwer zu erkennen. Die gesamtheitliche Betrachtung aller Funktionsbeziehungen führt daher nicht unmittelbar zu einem besseren Verständnis des Zusammenwirkens aller Funktionen.

Abbildung 6.23: Beziehungen eines bestimmten Typs, hier prepare

Auf der Grundlage eines Beziehungskalküls können und sollten grundsätzlich alle Funktionsbeziehungen in die Berechnung abgeleiteter Beziehungen einfließen. Ohne eine geeignete Werkzeugunterstützung ist aber zum Beispiel die Verifikation eines Beziehungsmodells kaum vorstellbar. Aber abgesehen von den sicherlich noch benötigten Verifikationsmethoden bietet eine Nutzungsarchitektur bereits eine gebührende Unterstützung für den Systementwurf. Wie in den vorhergehenden Abschnitten gesehen, legen die analysierten Zusammenhänge zwischen den Nutzungsfunktionen bereits ein entsprechendes Design der Schnittstellen nahe. Darüber hinaus helfen die festgelegten Funktionsbeziehungen auch ohne Werkzeug, das Wirkungsgefüge eines Softwaresystems zu verstehen. Dafür ist

es allerdings notwendig, sich auf bestimmte Aspekte zu konzentrieren. Im Grunde kann die Nutzungssicht selbst auch durch verschiedene Perspektiven strukturiert werden. Eine naheliegende Perspektive haben wir bereits im Rahmen des Abschnitts 6.1 kennen gelernt, und zwar als Funktionsnetz mit der Strukturierung durch Superfunktionen und Subfunktionen.

Es bietet sich auch an, nur Funktionsbeziehungen einer bestimmten Beziehungsklasse oder eines bestimmten Beziehungstyps zu beleuchten. Beispielsweise können im Zusammenhang mit der Identifizierung notwendiger Systemmodi alle exclude und prepare Beziehungen herangezogen werden. In Abbildung 6.23 ist der entsprechende Teil der Nutzungsarchitektur abgebildet. Die gestrichelt gezeichneten Verbindungen stellen dort wiederum die Subfunktionsrelation dar. Gerade im Hinblick auf die im vorhergehenden Abschnitt diskutierten impliziten Beziehungen sollten Subfunktionsbeziehungen stets miteinbezogen werden. Beispielsweise ist zwischen den Funktionen «Nummer wählen» und «Einschalten» zwar kein direkter prepare Zusammenhang formuliert. Dennoch werden diese Funktionen nur in unterschiedlichen Betriebsmodi zur Verfügung stehen, da «Telefonieren» eine Superfunktion von «Nummer wählen» ist und außerdem

‘*Einschalten*’ **prepares** ‘*Telefonieren*’

gilt.

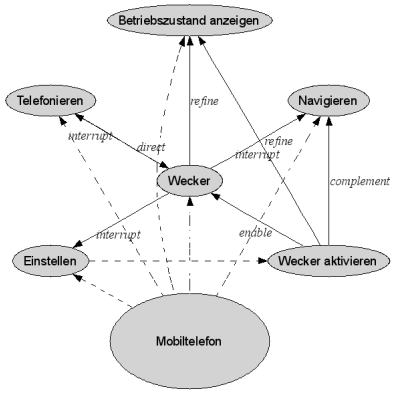


Abbildung 6.24: Beziehungskontext einer Nutzungsfunktion, hier «Wecker»

Eine andere sinnvolle Betrachtungsweise stellt eine Gruppe von Funktionen in einem bestimmten Anwendungsszenario dar. Solch eine Kontextsicht ist ebenfalls ein probates Mittel der Strukturierung. Nur Beziehungen eines speziell interessierenden Kontextes sind sichtbar und nicht Relevantes wird ausgeblendet. Zum Beispiel mag etwa im Falle einer beabsichtigten Änderungen an einer bestimmten Funktion interessieren, in welchen Kontext diese Funktion eingebettet ist. Sprich, welche Abhängigkeiten existieren und auf

welche Nutzungsfunktion wirkt sich eine Änderung gegebenenfalls aus. Als Beispiel sei hier in Abbildung 6.24 der Beziehungskontext der «Wecker» Funktion illustriert. Wie dort zu erkennen ist, wird sie unter anderem von der Funktion «Wecker aktivieren» kontrolliert und beeinflusst auf der anderen Seite die Ausgaben der Funktion «Betriebszustand anzeigen». Darüber hinaus kann sie die Funktionen «Telefonieren», «Einstellen» sowie «Navigieren» und damit insbesondere auch die Funktion «Wecker aktivieren» unterbrechen. Schließlich sind die möglichen Eingaben abhängig vom Zustand der «Telefonieren» Funktion.

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit liegt der Schwerpunkt auf der Erweiterung der Anforderungsanalyse. Durch die Möglichkeit des expliziten Spezifizierens von Funktionsbeziehungen im Rahmen der Entwicklung einer Nutzungsarchitektur wird der Analyseprozess verbessert und ein sanfterer Übergang hin zum Systementwurf geschaffen.

Ein Teil des Ergebnisses dieser Arbeit ist die umfangreiche Analyse klassischer und aktueller Konzepte im Bereich des Requirements Engineerings und des Entwurfs von Softwaresystemen, dienstbasierter Methoden sowie typischer Repräsentanten verschiedener Anwendungsdomänen. Daraus werden einige nützliche Beziehungstypen zur Beschreibung von Abhängigkeiten aus der Nutzungsperspektive abgeleitet. Diese werden ergänzt durch Beziehungstypen, deren Notwendigkeit sich aus praktischen Überlegungen insbesondere aufgrund der Fallstudie ergibt.

Die Strukturierung der Blackboxsicht eines Systems wird, wie oft bei Systemstrukturierungen klassischerweise angewandt, mithilfe unterschiedlicher Sichten erreicht. Daraus resultiert die Definition verschiedener Funktionsbeziehungen, die sich ihrerseits überlagern können oder aber sich auch gegenseitig ausschließen.

Den Schwerpunkt der Arbeit bildet die Definition einer praktikablen Menge von Beziehungstypen. Damit ist es möglich, grundlegende Zusammenhänge von Nutzungsfunktionen zu beschreiben. Mit ihnen können sowohl Funktionshierarchien gebildet werden, als auch die Beeinflussung von Funktionen dargestellt werden. Entscheidend für ein fehlerfreies Zusammenspiel der Funktionen ist aber deren Kombinierbarkeit. Mit den vorgestellten Beziehungstypen lässt sich diese nun explizit beschreiben. Durch die so erreichte Strukturierung der Nutzungsfunktionen eines Systems - also der funktionalen Eigenschaften aus Nutzungssicht - wird gleichsam dessen Blackboxsicht strukturiert.

Eine Fallstudie anhand konkret spezifizierter Funktionen eines Mobiltelefons, sowie die

Spezifikation der Zusammenhänge dieser Funktionen als konkrete Nutzungsarchitektur, runden die Arbeit ab.

Ein Vergleich mit DFC

Aufgrund der unterschiedlichen Zielsetzungen können die in Kapitel 2 untersuchten Ansätze nur bedingt den Konzepten dieser Arbeit sinnvoll gegenübergestellt werden. Wir greifen uns an dieser Stelle dennoch die „Distributed Feature Composition“ (DFC) heraus, da sie zumindest hinsichtlich des Kontextes – Anforderungsanalyse, Nutzungsfunktionen und Beeinflussung von Funktionen – vergleichbar erscheint.

Eine DFC Architektur besteht prinzipiell aus einer beliebigen Anzahl von Kommunikationseinheiten. Sie umfasst im Wesentlichen stets eine Grundfunktion für den einfachen Verbindungsaufbau sowie eine Menge von „Features“, die durch Subskribierung dynamisch hinzugefügt werden und jeweils eine gewünschte Funktionalität abdecken. Durch die Hinzunahme eines Features wird in der Regel das Gesamtverhalten des Systems modifiziert. Da die syntaktische Schnittstelle eines Features im Grunde vorgegeben ist, lassen sich Features zumindest aus syntaktischer Perspektive beliebig kombinieren.

Die Nutzungsfunktionen (Features) werden zunächst isoliert beschrieben und als Spezifikationen in ein ausführbares Modell integriert[1]. Das Modell ist bereits sehr stark angelehnt an die Architektur eines Telekommunikations-Switches, bei dem sich Nutzer für Features registrieren, die durch eine zentrale Steuerungseinheit (Router) ausgewählt werden. Die Features selbst werden als unterschiedliche, so genannte „Boxes“, behandelt und sind dort in einer Art Pipe & Filter Architektur hintereinander geschaltet. Funktionen kommunizieren dort nicht auf direktem Wege. Der Austausch der Nachrichten erfolgt stets über einen Router, der damit Funktionen auch „umgehen“ kann.

Das Prinzip für die Analyse von Zusammenhängen besteht nun darin, dass die unabhängig voneinander spezifizierten Features in verschiedenen Kombinationen ausprobiert werden. Als Ergebnis einer Simulation erhält man ein bestimmtes Systemverhalten, welches es dann zu beurteilen gilt. Sofern das simulierte Verhalten dem gewünschten Verhalten entspricht, gibt es keine beziehungsweise nur erwartete „feature interactions“. Sonst, also bei „unwanted feature interaction“, kommen aufgrund der vorgegebenen architekturellen Bedingungen nur folgende Maßnahmen in Betracht, das gewünschte Gesamtverhalten des Systems zu erreichen:

- Änderung der Spezifikation beziehungsweise Umprogrammieren einzelner Features, mit dem Ziel, den für Telekommunikationsdienste spezifischen Interaktionstyp der Features („hide“, „spoof“, „cancel“, „retarget“ [Zav01]) zu ändern.
- Definition einer partiellen Ordnung der Features, die von dem Router forciert wird. Das heißt, es gibt eine vorgeschriebene und damit keine beliebige Reihenfolge der Ausführung von Features.

[1] Ausführbare Spezifikationen in Promela und Z [Zav99b].

DFC ist sehr stark auf die Modellierung von Telekommunikationsdiensten zugeschnitten. Die gewählte Pipe & Filter Architektur (über Router) mit Subskribierung von Features ist dort angebracht und für diese Anwendungsdomäne ausreichend [BCP+04]. Die Zusammenhänge der Funktionen beschränken sich im Wesentlichen auf die Erweiterung von Funktionen (Grundfunktion + Feature + ...) und der Manipulation des Ausgabeverhaltens durch die Definition einer partiellen Ordnung der Features. Die Übertragung von DFC in andere Anwendungsdomänen ist alleine schon deshalb nicht einfach.

Davon abgesehen, gibt es auch bei der DFC keine explizite Charakterisierung der Funktionszusammenhänge aus Nutzungssicht. Das gewünschte Verhalten von kombinierten Funktionen wird nicht explizit aus Nutzungssicht spezifiziert, da a priori die Unabhängigkeit der Features angenommen wird. Ein Zusammenhang zwischen Funktionen wird erst über simulierte Kombinationsszenarien sichtbar.

Überhaupt ist bei der DFC keine besondere Strukturierung der Funktionen vorgesehen. Gegebenenfalls können zwar, wie bereits erwähnt, Funktionen partiell geordnet werden, allerdings nicht im Sinne einer Hierarchie von Funktionen. Daher ist auch die Kombinierbarkeit von Funktionen dort nur schwer nachzuvollziehen. Nicht zuletzt macht es aber gerade dies schwierig, Funktionszusammenhänge in DFC explizit zu formulieren, die über die oben genannten Interaktionstypen hinaus gehen. Im Grunde kann man zwar, ähnlich wie in dieser Arbeit dargestellt, auch bei der DFC unterscheiden zwischen kombinierbaren, bedingt kombinierbaren und nicht kombinierbaren Funktionen. In DFC hängt die Kombinierbarkeit beziehungsweise die bedingte Kombinierbarkeit von Funktionen nur damit zusammen, ob mögliche Interaktionen zufrieden stellend sind beziehungsweise erst mit eingeschränkter Ordnung der Funktionen akzeptabel sind. Diese Arbeit dagegen stützt sich auf den Zugang zu Funktionen im Sinne einer Verfeinerung des Schnittstellenverhaltens ab. Zusammenhänge zwischen Funktionen können damit viel differenzierter bewertet werden.

DFC reiht sich in die Gruppe der ausführbaren Modellierungsumgebungen ein mit dem Fokus auf Diensten, die isoliert voneinander spezifiziert werden, um dann mithilfe von Simulationsläufen potenzielle Feature Interactions der Dienste zu entdecken. Eine Abstraktion von architekturellen Gegebenheiten und spezifizierten Schnittstellenverhalten der Funktionen ist hier nicht vorgesehen. Die Konzepte der vorliegende Arbeit bieten dieses Abstraktionsniveau. Der Zusammenhang der Nutzungsfunktionen kann losgelöst von möglichen Architekturen und Schnittstellen spezifiziert werden und als wesentlicher Bestandteil der funktionalen Anforderungen an das System – im Sinne des gewünschten Verhaltens der kombinierten Funktionen – mit einfließen.

Weiterführende Arbeiten

Die Modellierung von Beziehungen zwischen Nutzungsfunktionen stellt eine konsequente Erweiterung der Anforderungsanalyse dar. Im nächsten Schritt sollte daher eine entsprechende prototypische Modellierungsumgebung zur Verfügung gestellt werden, um dafür

rasch eine Werkzeugunterstützung anzubieten. Als hervorragende Grundlage eignet sich hierfür etwa AutoRaid, da es bereits in das CASE Werkzeug AutoFocus 2 integriert ist.

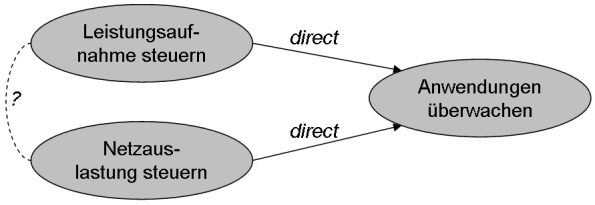
Die in Kapitel 5 diskutierten Beziehungstypen sind auf einem vergleichsweise hohen Abstraktionsniveau. Zwar wird auch dort bereits der Einfluss einer Funktion auf das Verhalten einer anderen Funktion differenziert betrachtet (zum Beispiel die Manipulation des Zugangs, das Ersetzen von Ausgaben oder die Störung von Funktionen). Man kann sich jedoch vorstellen, dass oft speziellere Beziehungstypen wünschenswert sind, um etwa Interaktionsschnittstellen der betroffenen Funktionen unmittelbar ableiten zu können. Dies führte gewissermaßen zu hierarchischen Zusammenhängen der Beziehungstypen selbst. Falls sich darüber hinaus der Zusammenhang zwischen zwei Funktionen nicht mit nur einem Beziehungstyp beschreiben lässt, sind die notwendigen Beziehungstypen sorgfältig auszuwählen. Eine in diesem Kontext interessante Fragestellung ist die Verträglichkeit von Beziehungstypen.

Unter der Voraussetzung, dass Begriffe wie die Verfeinerung oder die Verträglichkeit von Beziehungstypen formal fundiert sind, ließe sich eine Taxonomie von Beziehungstypen erstellen. Diese könnte im Rahmen einer systematischen, werkzeugbasierten Entwicklung einer Nutzungsarchitektur im Sinne einer methodischen Unterstützung ausgenutzt werden. Die Konsequenz einer Beziehung zwischen Funktionen kann die Notwendigkeit einer Interaktionsschnittstelle sein. Ab einem bestimmten Detaillierungsgrad eines Beziehungstyps ist es denkbar, Spezifikationsmuster für Interaktionsschnittstellen für die in Beziehung stehenden Nutzungsfunktionen anzubieten, die zur Designzeit instanziiert werden.

Für die Widerspruchsfreiheit des Beziehungsmodells ist die Formulierung von Verträglichkeitsbedingungen und die Prüfung der Verträglichkeit gegebener Beziehungstypen nur ein erster Schritt. Weitergehende Konsistenzprüfungen erfordern die Überprüfung von abgeleiteten Abhängigkeiten. Dazu ist die Definition eines umfangreichen Beziehungskalküls erforderlich. Interessant ist in diesem Zusammenhang insbesondere die Möglichkeit komplexer Anfragen, wie „In welchem Zusammenhang stehen die Funktionen f und g?“ oder „Sind die Funktionen f und g kombinierbar, wenn **affect**(h, g) gilt?“.

Rufen wir uns noch einmal die Situation des Beispiels auf Seite 5 in Erinnerung. Angenommen es gibt dort neben den beschriebenen Funktionen «Leistungsaufnahme steuern» und «Netzauslastung steuern» noch eine Funktion «Anwendungen überwachen». Diese soll notwendige Anwendungen starten oder fortsetzen, solange die Anzahl der laufenden Anwendungen kleiner als eine untere Schranke ist. Darüber hinaus soll sie nicht unbedingt notwendige Anwendungen stoppen, solange die Anzahl der laufenden Anwendungen noch größer als eine obere Schranke ist. Wenn nun beide Funktionen etwa zugleich in einer direct Beziehung zur Funktion «Anwendung überwachen» stehen, zum Beispiel wird von den Funktionen jeweils der Wert der unteren beziehungsweise der oberen Schranke gesetzt,

so weist dies auf eine kritische Abhängigkeit hin.



Solche versteckten Abhängigkeiten, die möglicherweise kritisch oder sogar unzulässig sind, sollten im Zusammenhang mit einem Beziehungskalkül aufzudecken sein. Eine Lösung kann hier auch tatsächlich die Einführung einer Beziehung sein, die in der Grafik mit einem Fragezeichen markiert ist, wie zum Beispiel eine disable Beziehung oder auch eine weitere direct Beziehung.

Verifikationsmechanismen von gegebenen Beziehungen und architektureller Umsetzung erfordern die Definition einer Abbildung der Nutzungssicht auf die logische Komponentenarchitektur (Komponenten und Interaktionsschnittstellen der Komponenten). Ob nun gegebene Beziehungen in einer Komponentenarchitektur korrekt umgesetzt sind, sollte anhand der Eigenschaften der Beziehungstypen verifizierbar sein. Die Umsetzung dieser Überprüfung ist elementar für die Unterstützung eines integrierten Analyse- und Designprozesses. Ungleich schwieriger wird es jedoch werden, bereits bei der Definition einer Beziehung geeignete Modellelemente einer Komponentenarchitektur zu generieren.

Eine letzte interessante Fragestellung ist auch die Möglichkeit der Generierung eines Beziehungsmodells aus einer gegebenen Komponentenarchitektur heraus. Als Unterstützung für ein verbessertes Systemverständnis ist dies ohne Zweifel sehr hilfreich. Dies erforderte allerdings, dass die Elemente der Komponentenarchitektur mit den funktionalen Anforderungen bereits in Beziehung gesetzt sind.

Literaturverzeichnis

[ADG98] Allen, Robert, Remi Douence und David Garlan: *Specifying and Analyzing Dynamic Software Architectures.* In: *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98)*, Lissabon, Portugal, März 1998. Eine erweiterte Version des Papiers „Specifying Dynamism in Software Architectures", das im September 1997 erschienen ist in den Proceedings of the Workshop on Foundations of Component-Based Software Engineering.

[AF205] *Web-Seite des Projekts* AutoFocus 2, 2005. `http://www4.in.tum.de/~af2`.

[All97] Allen, Robert: *A Formal Approach to Software Architecture.* Doktorarbeit, Carnegie Mellon, School of Computer Science, Januar 1997. Veröffentlicht als CMU Technical Report CMU-CS-97-144.

[ATT05] *Graphviz - Graph Visualization Software.* AT&T Corp., 2005. `http://www.graphviz.org/`.

[Bau90] Baumgarten, Bernd: *Petri-Netze, Grundlagen und Anwendungen.* Bibliographisches Institut, Mannheim, 1990.

[BCP+04] Bond, Gregory W., Eric Cheung, K. Hal Purdy, Pamela Zave und J. Christopher Ramming: *An Open Architecture For Next–Generation Telecommunication Services.* In: *ACM Transactions on Internet Technology IV(1)*, Februar 2004.

[Ber04a] Berenbach, Brian: *The Evaluation of Large, Complex UML Analysis and Design Model.* In: *26th International Conference on Software Engineering, ICSE*, Seiten 232–241, 2004.

[Ber04b] Berenbach, Brian: *Towards a unified model for requirements engineering.* In: *Fourth International Workshop on Adoption-Centric Software Engineering, ACSE*, Seiten 26–29, 2004.

[BR01] Braun, Peter und Martin Rappl: *Abstraction levels of embedded systems.* OMER-2: Workshop on Object-oriented Modeling of Embedded RT-Systems, Herrsching, Mai 2001.

[BR02] Braun, Peter und Martin Rappl: *A Model Based Approach for Automotive Software Development.* In: *OMER – Object-oriented Modeling of Embedded Real-Time Systems, LNI P-5.* Springer-Verlag, 2002.

[Bro03a] Broy, Manfred: *Modeling Services and Layered Architectures.* In: König, H., M. Heiner und A. Wolisz (Herausgeber): *Formal Techniques for Networked and Distributed Systems*, Band 2767 der Reihe *Lecture Notes in Computer Science*, Seiten 48–61. Springer, 2003.

[Bro03b] Broy, Manfred: *Multi-view Modeling of Software Systems*, 2003. Keynote. FM2003 Satellite Workshop on Formal Aspects of Component Software, 8–9 September, Pisa, Italien.

[Bro04a] Broy, Manfred: *Modellbasierung im Software Lifecycle – Stand und Potentiale*, Mai 2004.

[Bro04b] Broy, Manfred: *Software Engineering: Software im Automobil – Einführung und Überblick.* Vorlesung im SS04, Technische Universität München, 2004.

[Bro04c] Broy, Manfred: *Software Engineering: Software im Automobil – Eingebettete Systeme.* Vorlesung im SS04, Technische Universität München, 2004.

[Bro04d] Broy, Manfred: *Strukturiertes Requirements Engineering: Modellbasierte Anforderungsbeschreibung*, 2004. Keynote. 3. Requirements Engineering Tagung 2004, 8–10 März, München.

[Bro05] Broy, Manfred: *Service-oriented Systems Engineering: Specification and Design of Services and Layered Architectures – The Janus-Approach.* In: Broy, Manfred, Johannes Grünbauer, David Harel und Tony Hoare (Herausgeber): *Engineering Theories of Software Intensive Systems*, Nummer 195 in *Proceedings of the NATO Advanced Study Institute on Engineering Theories of Software Intensive Systems*, Marktoberdorf, Juni 2005. Kluwer Academic Publishers.

[BS01] Broy, Manfred und Ketil Stølen: *Specification and Development of Interactive Systems:* Focus *on Streams, Interfaces and Refinement.* Springer-Verlag, 2001.

[BSDV97] Bertram, T., W. Schröder, P. Dominke und A. Volkart: *CARTRONIC – ein Ordnungskonzept für die Steuerungs- und Regelungssysteme in Kraftfahrzeugen.* Systemengineering in der KFZ-Entwicklung S. 369–398, VDI-Berichte 1374, 1997.

[CE00] Czarnecki, Krzysztof und Ulrich W. Eisenecker: *Generative Programming – Methods, Tools, and Applications.* Addison-Wesley, 2000.

[CK04] Cengarle, María Victoria und Alexander Knapp: *UML 2.0 Interactions: Semantics and Refinement.* In: Jürjens, Jan, Eduardo B. Fernandez, Robert France und Bernhard Rumpe (Herausgeber): *3rd Int. Wsh. Critical Systems Development with UML (CSDUML'04)*, Seiten 85–99. Institut für Informatik, Technische Universität München, 2004. Technical Report TUM-I0415.

[CWG04] Cengarle, María Victoria, Stefan Wagner und Peter Graubmann: *From Feature Models to Variation Representation in MSCs.* In: Bosch, Jan (Herausgeber): *2nd Groningen Workshop on Software Variability Management (SVM'04)*, Technical Report IWI preprint 2004-7-01, Seiten 49–60. Instituut voor Wiskunde en Informatica, Rijksuniversiteit Groningen, 2004.

[DAC98] Dwyer, Matthew B., George S. Avrunin und James C. Corbett: *Property Specification Patterns for Finite-State Verification.* In: Ardis, Mark (Herausgeber): *Proc. 2nd Workshop on Formal Methods in Software Practice (FMSP-98)*, Seiten 7–15, New York, März 1998. ACM Press.

[DAC99] Dwyer, Matthew B., George S. Avrunin und James C. Corbett: *Patterns in Property Specifications for Finite-state Verification.* In: *Proceedings of the 21st International Conference on Software Engineering*, Mai 1999.

[DGH+05] Deubler, Martin, Johannes Grünbauer, Andreas Holzbach, Gerhard Popp und Guido Wimmel: *Kontextadaptivität in dienstbasierten Softwaresystemen.* Technischer Bericht TUM-I0511, Institut für Informatik, Technische Universität München, Juli 2005.

[DGJW04] Deubler, Martin, Johannes Grünbauer, Jan Jürjens und Guido Wimmel: *Sound Development of Secure Service based Systems.* In: *2nd*

International Conference on Service Oriented Computing (ICSOC), New York, 15. – 18. November 2004.

[DGP+04a] Deubler, Martin, Johannes Grünbauer, Gerhard Popp, Guido Wimmel und Christian Salzmann: *Tool Supported Development of Service Based Systems.* In: *11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, Busan, Korea, 30. November – 3. Dezember 2004. IEEE Computer Society.

[DGP+04b] Deubler, Martin, Johannes Grünbauer, Gerhard Popp, Guido Wimmel und Christian Salzmann: *Towards a Model-Based and Incremental Development Process for Service-Based Systems.* In: Hamaza, M. H. (Herausgeber): *Proceedings of the IASTED International Conference on Software Engineering (IASTED SE 2004)*, Seiten 183–188, Innsbruck, Austria, 17. – 19. Februar 2004.

[DKMR05] Deubler, Martin, Ingolf Krüger, Michael Meisinger und Sabine Rittmann: *Modeling Crosscutting Services with UML Sequence Diagrams.* In: *8th International Conference on Model Driven Engineering Languages and Systems (MoDELS'05)*, Jamaica, 2. – 7. Oktober 2005. ACM / IEEE.

[DOO05] *Homepage Telelogic AB*, 2005. `http://www.telelogic.de/`.

[GDMR04] Gnatz, Michael, Martin Deubler, Michael Meisinger und Andreas Rausch: *Towards an Integration of Process Modeling and Project Planning.* In: *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004), ICSE 2004*, Mai 2004.

[Gen04] Genz, Henning: *Wie die Naturgesetze Wirklichkeit schaffen.* Rowohlt, 2004.

[GKN02] Gansner, Emden R., Eleftherios Koutsofios und Stephen C. North: *Drawing graphs with* dot. AT&T Corp., Februar 2002. `http://www.graphviz.org/Documentation/dotguide.pdf`.

[GKNV93] Gansner, Emden R., Eleftherios Koutsofios, Stephen C. North und Kiem-Phong Vo: *A Technique for Drawing Directed Graphs.* IEEE Transactions on Software Engineering, 19(3):214–230, Mai 1993.

[Grü08] Grünbauer, Johannes: *Feature Interactions auf Nutzungsebene – Modellierung und Analyse der Abhängigkeiten.* Doktorarbeit, Technische Universität München, 2008.

[HL04] Hauser, Tobias und Ulrich M. Löwer: *Web Services – Die Standards.* Galileo Press, 2004. `http://www.web-services.cc`.

[HMR+98] Huber, Franz, Sascha Molterer, Andreas Rausch, Bernhard Schätz, Mark Sihling und Oskar Slotosch: *Tool supported Specification and Simulation of Distributed Systems.* In: *International Symposium on Software Engineering for Parallel and Distributed Systems*, Seiten 155–164, 1998.

[Hoa85] Hoare, C. A. R.: *Communicating Sequential Processes.* Prentice-Hall, International Series in Computer Science, 1985.

[HPR97] Holzmann, Gerard J., Doron A. Peled und Margaret H. Redberg: *Design Tools for Requirements Engineering.* Bell Labs Technical Journal, Vol. 2(Nr. 1):Seiten 86–95, 1997.

[ITU99] *ITU-T Recommendation Z.120 – Message Sequence Chart (MSC).* Genf, 1999.

[Jen98] Jensen, Kurt: *An Introduction to the Practical Use of Coloured Petri Nets.* Lecture Notes in Computer Science: Lectures on Petri Nets II: Applications, 1492, 1998.

[JIN05] *Jini Community*, 2005. `http://www.jini.org`.

[JZ03] Jackson, Michael und Pamela Zave: *The DFC Manual*, 2003.

[KCH+90] Kang, Kyo C., Sholom G. Cohen, James A. Hess, William E. Novak und A. Spencer Peterson: *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technischer Bericht CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, November 1990.

[KCJ98] Kristensen, Lars M., Soren Christensen und Kurt Jensen: *The practitioner's guide to coloured Petri nets.* International Journal on Software Tools for Technology Transfer: Special section on coloured Petri nets, 2(2):98–132, 1998. available at http://sttt.cs.uni-dortmund.de/.

[KFKK03] Kustosch, Mario, Rasmus Frei, Werner Kind und Rainer Kallenbach: *Flexible Markencharakterisierung mit der Cartronic-Systemarchitektur.* AUTO & ELEKTRONIK, Januar 2003.

[KGM+04] Krüger, Ingolf H., Diwaker Gupta, Reena Mathew, Praveen Moorthy, Walter Phillips, Sabine Rittmann und Jaswinder Ahluwalia: *Towards a Process and Tool-Chain for Service-Oriented Automotive Software Engineering.* In: *Proceedings of the ICSE 2004 Workshop on Software Engineering for Automotive Systems (SEAS)*, 2004.

[KKL+98] Kang, Kyo C., Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin und Moonhang Huh: *FORM: A feature-oriented reuse method with domain-specific reference architectures.* Annals of Software Engineering, 5:143–168, 1998.

[KM04] Krüger, Ingolf H. und Reena Mathew: *Systematic Development and Exploration of Service–Oriented Software Architectures.* In: *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, 2004.

[Krü04] Krüger, Ingolf H.: *Service Specification with MSCs and Roles.* In: Hamaza, M. H. (Herausgeber): *Proceedings of the IASTED International Conference on Software Engineering (IASTED SE 2004)*, Seiten 42–47, Innsbruck, Austria, 17. – 19. Februar 2004.

[LDW04] Leupers, Rainer, Malte Dörper und Andreas Wieferink: *Chips mit System.* c't Magazin, 20:92–99, September 2004.

[LFS+01] Lapp, A., P. Torre Flores, J. Schirmer, D. Kraft, W. Hermsen, T. Bertram und J. Petersen: *Softwareentwicklung für Steuergeräte im Systemverbund - Von der CARTRONIC-Domänenstruktur zum Steuergerätecode.* VDI-Gesellschaft „Fahrzeug- und Verkehrstechnik“ S. 249–276, VDI-Berichte 1646 „Elektronik im Kraftfahrzeug“, 2001.

[LTX01] Lorentsen, Louise, Antti Pekka Tuovinen und Jianli Xu: *Modeling Feature Interactions in Mobile Phones.* In: *ECOOP Workshop – Feature Interaction in Composed Systems*, 2001.

[Luc96] Luckham, David C.: *Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events.* DIMACS Partial Order Methods Workshop IV, Princeton University, August 1996. `ftp://pavg.stanford.edu/pub/Rapide-1.0/pom.ps.Z`.

[Mar03] Martin, Robert C.: *UML for Java Programmers.* Prentice Hall, 2003.

[MEW05] *Web-Seite des Forschungsprojekts MEWADIS Modellbasierte Entwicklung adaptiver Dienste*, 2005. `http://www4.in.tum.de/~mewadis`.

[MK96] Magee, Jeff und Jeff Kramer: *Dynamic structure in software architectures.* In: *Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Seiten 3–14, San Fransisco, Oktober 1996.

[MRD+04] Meisinger, Michael, Andreas Rausch, Martin Deubler, Michael Gnatz, Ulrike Hammerschall, Inga Küffer und Sascha Vogel: *Das V Modell 200x - ein modulares Vorgehensmodell*. In: Ralf Kneuper, Roland Petrasch, Manuela Wiemers (Herausgeber): *11. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zur Akzeptanz von Vorgehensmodellen*. Shaker Verlag, April 2004.

[MTK06] *MTK - Mobility Toolkit*, 2006. Früher SMTK (Siemens Mobility Toolkit), heute auf dem BenQ *mobile* Developer Portal `http://www.benq-siemens.com/developer`.

[Nat93] National Institute of Standards and Technology (NIST): *Draft Federal Information Processing Standards Publication 183, Standard for integration definition for funczion modeling (IDEF0)*, 1993.

[NP03] Nelson, E. C. und K. V. Prasad: *Automotive Infotronics: An emerging domain for Service-Based Architecture*. In: Broy, Manfred, Heinrich Hussmann, Ingolf Krüger und Bernhard Schätz (Herausgeber): *Service-Based Software Engineering Workshop, Pisa*, Seiten 3–14, September 2003.

[OHE94] Orfali, Robert, Dan Harkey und Jeri Edwards: *Essential client/server survival guide*. Wiley & Sons Inc., 1994.

[OHE96] Orfali, Robert, Dan Harkey und Jeri Edwards: *The Essential Distributed Objects Survival Guide*. Wiley & Sons Inc., 1996.

[OMG04] *Unified Modeling Language 2.0 Superstructure Final Adopted Specification*, August 2004. Object Management Group Document ptc/03-08-02.

[OMG05] *Web-Seite der Object Management Group*, 2005. `http://www.omg.org/`.

[OSG03] *OSGi™ Service Platform Specification*. Open Services Gateway Initiative, März 2003. Release 3, `http://www.osgi.org`.

[Par98] Partsch, Helmuth: *Requirements-Engineering systematisch – Modellbildung für softwaregestützte Systeme*. Springer-Verlag, 1998.

[PSC+01] Pulvermüller, Elke, Andreas Speck, James O. Coplien, Maja D'Hondt und Wolfgang DeMeuter: *Feature Interaction in Composed Systems*. In: *ECOOP Workshop – Feature Interaction in Composed Systems*, 2001.

[RAI05] *Web-Seite des Projekts AutoRAID „AutoFocus Requirements Analysis Integrating Development“*, 2005. `http://www4.in.tum.de/~autoraid/`.

[Rit04] Rittmann, Sabine: *Exploring Service–Oriented Software Development for Automotive Systems.* Diplomarbeit, Technische Universität München, 2004.

[Rit08] Rittmann, Sabine: *A methodology for modeling usage behavior of multi-functional systems.* Doktorarbeit, Technische Universität München, 2008.

[RS01] Rumpe, Bernhard und Robert Sandner: *UML – Unified Modeling Language im Einsatz.* Automatisierungstechnik, 49, September – November 2001. Oldenbourg Verlag.

[Sal02] Salzmann, Chris: *Modellbasierter Entwurf spontaner Komponentensysteme.* Doktorarbeit, Technische Universität München, 2002.

[Sch04] Schinarakis, Kosta: *Programmiertes Problem.* Süddeutsche Zeitung, WIRTSCHAFT(Nr. 199):20, August 2004.

[SDZ96] Shaw, Mary, Robert DeLine und Gregory Zelesnik: *Abstractions and Implementations for Architectural Connections.* In: *3rd Int. Conf. on Configurable Distributed Systems*, Mai 1996.

[SEF04] Shehata, Mohamed, Armin Eberlein und Abraham Fapojuwo: *The Use of Semi-Formal Methods for Detecting Requirements Interactions.* In: Hamaza, M. H. (Herausgeber): *Proceedings of the IASTED International Conference on Software Engineering (IASTED SE 2004)*, Seiten 230–235, Innsbruck, Austria, 17. – 19. Februar 2004.

[Slo98] Slotosch, Oskar: *Quest: Overview over the Project.* In: Hutter, D., W. Stephan, P Traverso und M. Ullmann (Herausgeber): *Applied Formal Methods – FM-Trends 98*, Seiten 346–350. Springer LNCS 1641, 1998.

[SS03a] Salzmann, Chris und Bernhard Schätz: *Service Based Software Specification.* In: ETAPS (Herausgeber): *Proceedings of International Workshop on Test and Analysis of Component Based Systems (TACOS)*, Warschau, Polen, 2003.

[SS03b] Schätz, Bernhard und Christian Salzmann: *Service-Based Systems Engineering: Consistent Combination of Services.* In: *Proceedings of ICFEM 2003, Fifth International Conference on Formal Engineering Methods.* Springer LNCS 2885, 2003.

[STK02] Snell, J., D. Tidwell und P. Kulchenko: *Programming Web Services with SOAP.* O'Reilly, 2002.

[Sys05] *Systems Modeling Language (SysML) Specification*, Januar 2005. Version 0.9 (Draft), `http://www.sysml.org`.

[TAC+03] Thatte, S., T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, I. Trickovic und S. Weerawarana: *Business Process Execution Language for Web Services „BPEL4WS Spezifikation“*. BEA, IBM, Microsoft, SAP AG, Siebel Systems, Version 1.1 Auflage, Mai 2003. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`.

[Tan89] Tanenbaum, Andrew S.: *Computer Networks*. Prentice-Hall Inc., 2. Auflage, 1989.

[UPP05] *UPnP Forum*, 2005. `http://www.upnp.org`.

[vdBBRS02] Beeck, Michael von der, Peter Braun, Martin Rappl und Christian Schröder: *Automotive Software Development: A Model Based Approach*. In: *World Congress of Automotive Engineers, SAE Technical Paper Series 2002-01-0875*, 2002.

[vdML02] Massen, Thomas von der und Horst Lichter: *Modeling Variability by UML Use Case Diagrams*. In: *Proc. Int. Workshop on Requirements Engineering for Product Lines*, 2002. Technical Report: ALR-2002-033.

[VMX05] *Web-Seite des neuen V-Modell^{TM} XT - Der Entwicklungsstandard für IT-Systeme des Bundes*, 2005. `http://www.v-modell-xt.de/`.

[WCG04] Wagner, Stefan, María Victoria Cengarle und Peter Graubmann: *Modelling System Families with Message Sequence Charts: A Case Study*. Technischer Bericht TUM-I0416, Institut für Informatik, Technische Universität München, 2004.

[Zav99a] Zave, Pamela: *FAQ Sheet on Feature Interaction*. AT&T, 1999. `http://www.research.att.com/~pamela/faq.html`.

[Zav99b] Zave, Pamela: *Formal Description of Telecommunication Services in Promela and Z*. In: Broy, M. und R. Steinbrüggen (Herausgeber): *Calculational System Design (Proceedings of the Nineteenth International NATO Summer School)*, Seiten 395–420. IOS Press, Amsterdam, 1999.

[Zav01] Zave, Pamela: *An experiment in feature engineering*. In: *Essays by the Members of the IFIP Working Group on Programming Methodology*. Springer, 2001.