Electronic version of an article published in **Schnieder, E. (Hrsg) Entwurf komplexer Automatisierungssysteme. Entwurfsmethodik, Modellbildung, Werkzeuge und Anwendungen. 8. Fachtagung, 11. bis 13. Juni 2003 in Braunschweig (EKA'03); Braunschweig: TU Braunschweig, Institut für Verkehrssicherheit und Automatisierungstechnik; pp. 317-339**

Beschreibung elektronischer Steuergeräte mit Use Cases und Statecharts

Christian Denger, Barbara Paech, Antje von Knethen, Daniel Kerkow
Fraunhofer Institut Experimentelles Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern
Tel.: +49 631 707 0
Fax: +49 631 707 200
E-Mail: {denger,paech,knethen,kerkow}@iese.fhg.de

Abstract: In der Automobilentwicklung nimmt die Entwicklung von Software einen immer höheren Stellenwert ein. Die Komplexität der Funktionalität dieser Software erfordert insbesondere einen auf die Bedürfnisse der Beteiligten zugeschnittenen Anforderungsprozess. Use Cases und Statecharts haben sich inzwischen bei vielen Anwendungen als geeignete Anforderungsbeschreibungstechniken durchgesetzt. In diesem Beitrag zeigen wir anhand eines konkreten Fallbeispiels aus der Praxis, wie eine Kombination von Use Cases und Statecharts den speziellen Erfordernissen der Anforderungsbeschreibung von Steuergeräten gerecht wird. Insbesondere motivieren wir die Eignung von Use Cases zur Beschreibung von funktionalen Anforderungen im Systemlastenheft und die Eignung von Statecharts zur Beschreibung im Systempflichtenheft. Detaillierte Richtlinien zur Erstellung der Beschreibungen unterstützen die praktische Anwendbarkeit dieses Ansatzes.

Stichworte: Anforderungsspezifikation, Use Cases, Statecharts

1 Einleitung

In der Automobilentwicklung nimmt die Entwicklung von Software auf Steuergeräten einen immer höheren Stellenwert ein. Moderne Fahrzeuge enthalten bis zu 80 Steuergeräte, komplexe Funktionen sind zunehmend Steuergerät-übergreifend. Wie in [WW02] beschrieben, erfordert die Komplexität der Funktionalität insbesondere einen auf die Bedürfnisse der Beteiligten zugeschnittenen Anforderungsprozess. Die Herausforderung dabei ist, geeignete Beschreibungsebenen und Repräsentationsformen festzulegen, damit Software- und Hardwareentwickler, Auftraggeber und Zulieferer effizient miteinander kommunizieren können.

Um diese Herausforderung anzugehen, hat das Fh IESE im BMBF-Projekt QUASAR eine Dokumentenstruktur für Anforderungen entwickelt. Die Struktur wurde basierend auf dem RE-KIT-FRAIME-Ansatz des Fh IESE zur Verbesserung von Requirements Engineering Prozessen

[PK00] entwickelt. Dazu wurden zunächst die Schwachstellen der existierenden Dokumente in Interviews mit Experten eines Automobilherstellers erhoben. Die Erhebung zeigte, dass

- System- und Softwareanforderungen klar getrennt werden sollten. Oft wird die Schnittstelle zwischen Hardware, Software und Mechanik nicht klar definiert und die Softwareentwickler haben keine klare Grundlage für ihre Entwicklungstätigkeiten.

- es nötig ist, den Kontext von Funktionen durch die Benutzersicht und die Abhängigkeit zwischen den Funktionen zu dokumentieren. Diese Information ist notwendig, um bei Änderungen und bei Wiederverwendung einzelner Funktionen zwischen verschiedenen Serien und Typen von Autos entscheiden zu können, welche anderen Funktionen durch die Änderungen beeinflusst sind und welche übergreifende Funktionalität bei der Wiederverwendung zu berücksichtigen ist.

- Lasten- und Pflichtenheft getrennt dokumentiert werden sollten, sodass das Lastenheft eine Grundlage für die Ausschreibung sein kann, in der u.a. nicht zuviel internes Know-how des Auftraggebers preisgegeben wird.

- Funktionen in der Anforderungsphase unabhängig von Steuergeräten zu beschreiben sind, um eine flexible Zuordnung der Funktionen zu den Steuergeräten zu einem späteren Zeitpunkt (ggf. sogar erst zur Laufzeit) zu ermöglichen.

Aufbauend auf den Interviews wurde eine Dokumentenstruktur für Anforderungen entwickelt, bestehend aus *Systemlastenheft und -pflichtenheft, sowie Softwarelastenheft und – pflichtenheft* [K+01]. Diese Struktur ermöglicht einen effizienten Umgang mit den Anforderungen, da sich die für bestimmte Gruppen wichtige Informationen in unterschiedlichen Dokumenten befindet. Um den Aufwand für die Dokumentation gering zu halten, werden alle Anforderungen in einem Requirements Management Werkzeug wie z.B. DOORS [DOORS] oder RequisitePro [RequisitePro] verwaltet und die Dokumente dann mit Hilfe der Filterfunktion dieser Werkzeuge generiert. Im folgenden verstehen wir unter *Dokument* deshalb *die für eine bestimmte Gruppe wichtige Menge von Anforderungen*. Aufgrund der Anforderungen dieser Gruppe wurden die Dokumenteninhalte (in Form von Templates) festgelegt. Dabei geht ein, wer diese Inhalte liefern kann, wer welche Entscheidungen aufgrund des Dokumentes trifft, und wer sich mit wem anhand der Dokumente abstimmt. Die Werkzeuge erlauben auch eine effiziente Unterstützung der Verfolgbarkeit zwischen Anforderungen, da Abhängigkeiten zwischen Anforderungen durch Links dokumentiert und dann effizient abgefragt werden können (z.B. durch Bilden der transitiven Hülle). Die Verfolgbarkeit unterstützt insbesondere Änderungen und Wiederverwendung, aber auch Konsistenzüberprüfung. Weiterhin wurden für jedes Dokument spezifische Beschreibungs- und Validierungstechniken (z.B. Checklisten für Inspektionen) festgelegt.

In diesem Beitrag konzentrieren wir uns auf die Beschreibung von *funktionalen Anforderungen* im Systemlasten- und pflichtenheft. Zur Beschreibung der funktionalen Anforderungen im Lastenheft werden *Use Cases (UCs),* im Pflichtenheft *Statecharts (SCs)* verwendet. Im weiteren werden wir diese Festlegung durch die Angabe von Kriterien an die Verhaltensbeschreibung in diesen Dokumenten begründen. Weiterhin geben wir anhand eines Beispiels einen Überblick über die in QUASAR entwickelten Richtlinien zur Erstellung von UCs und SCs für Steuergeräte. Die Richtlinien stellen sicher, dass die erstellten UCs und SCs die genannten Kriterien erfüllen. Sie bündeln unsere Erfahrungen und sollen den Anwendern eine effiziente und fehlerfreie Erstellung der Anforderungsdokumente ermöglichen. Die Richtlinien zur Erstellung der SCs nutzen die Struktur der UCs. Dadurch werden wir auch insbesondere die Unterschiede und Synergien zwischen den Beschreibungen deutlich machen.

Der Beitrag ist wie folgt strukturiert: In Abschnitt 2 stellen wir die Kriterien zur Beschreibung von Anforderungen im Lastenheft vor und geben Richtlinien zur Erstellung von UCs an, die zur Erfüllung dieser Kriterien beitragen. Abschnitt 3 behandelt in gleicher Weise SCs. Im letzten Abschnitt fassen wir noch mal die wesentlichen Schnittstellen zwischen UCs und SCs zusammen und diskutieren unsere Erfahrungen und andere Ansätze zur Verbindung von UCs und SCs. Weiterhin geben wir einen Ausblick auf weitere Forschungsarbeiten.

2 Beschreibung der funktionalen Anforderungen im Lastenheft mit Use Cases

Im folgenden stellen wir unsere Kriterien und Richtlinien zur Beschreibung des Systemlastenheftes mit Use Cases vor. Dabei ist wichtig festzuhalten, dass im Automobilbau das in einem Dokument beschriebene System ein einzelnes Steuergerät, aber auch den gesamten Innenraum umfassen kann [WW02]. Wir verwenden deshalb *System* im folgenden für eine in einem bestimmten Projekt wichtige Menge von Funktionalitäten.

2.1 Anforderungen an das Lastenheft

Das Lastenheft umfasst die Sicht des Auftraggebers auf das System. Es beschreibt, *was* das System leisten soll, aber nicht *wie* es das erreicht [VV]. Der Unterschied zwischen *wie* und *was* ist allerdings oft unscharf. Um präzise Vorgaben für den Inhalt eines Dokumentes machen zu können, ist es deshalb wichtig, zu berücksichtigen, wer es erstellt, wer damit arbeitet und welche Entscheidungen auf der Basis dieses Dokuments zu fällen sind.

Bei der Systementwicklung im Automobilbau wird das Lastenheft vom Autohersteller, insbesondere von den Marketing-Experten und Anforderungsingenieuren, erstellt. Es ist einerseits die Grundlage von Hersteller-internen Diskussionen darüber, welche Funktionalitäten (insbesondere Neuerungen) ein System (z.B. in einer neuen Autoserie) umfassen wird, andererseits ist es oft Grundlage für eine Ausschreibung. Damit dient es dem Zulieferer zu internen Diskussionen, ob und mit welchem Aufwand die beschriebenen Funktionalitäten

erbracht werden können. Dazu muss der Zulieferer die Hintergründe für bestimmte Anforderungen verstehen, d.h. es muss deutlich werden, welche Aufgaben der Benutzer, insbesondere des Fahrers, wie durch das System unterstützt werden.

Systeminterna sind nur wichtig, wenn sie zwingend vom Auftraggeber vorgeschrieben sind. Dies bedeutet insbesondere, dass von konkreten Sensoren und Aktuatoren soweit wie möglich abstrahiert wird. Wir verwenden deshalb im folgenden die Unterscheidungen des 4-Variablenmodells von Parnas [PM95]. Systemverhalten wird dabei durch die Abbildung von *monitorierten* Größen auf *kontrollierte* Größen beschrieben. Diese Größen abstrahieren von konkreten Sensoren und Aktuatoren. Letztere werden erst im Hardware-Entwurf festgelegt und sind dann Bestandteil der Softwarespezifikation. [vKn01] folgend unterscheiden wir bei den monitorierten und kontrollierten Größen weiterhin Größen, die nur *indirekt* beobachtbar oder beeinflussbar sind, und deshalb abgeleitet werden müssen (z.B. dass sich ein Objekt zwischen Fensteroberkante und Fensterrahmen befindet) von *atomaren* Größen, die direkt messbar sind (wie z.B., dass das Fenster sich nach oben bewegen sollte, aber keine Positionsveränderung wahrgenommen wird). Aus Benutzersicht sind vor allem die indirekten Größen wichtig.

Darüber hinaus ist eine gute Strukturierung von Anforderungsdokumenten zur Unterstützung von Wiederverwendbarkeit und Änderbarkeit wichtig. Da Anforderungen verschiedener Automobiltypen und – serien eines oder mehrerer Hersteller sich häufig überlappen, übernehmen die Entwickler oft Teile einer Beschreibung in eine neue [vKn+02].

2.2 Kriterien zur Verhaltensbeschreibung im Lastenheft

Aus den im letzten Abschnitt beschriebenen Anforderungen ergeben sich mehrere Kriterien, die die Beschreibung der Systemfunktionalität im Lastenheft erfüllen muss. Diese Kriterien sind im einzelnen:

> **(LH1)** Die Beschreibung muss möglichst allgemein verständlich die Benutzersicht auf die Systemfunktionen wiedergeben, um insbesondere die Kommunikation zwischen Management, Marketing und Benutzern zu unterstützen.
>
> **(LH2)** Es muss eine explizite Nennung aller Benutzeraufgaben und Systemverantwortlichkeiten erfolgen. Das Systemverhalten muss aber nicht im Detail beschrieben werden. Insbesondere sollte von internen technischen Problemen, die für den Benutzer nicht sichtbar sind, abstrahiert werden, wenn spezifische Lösungen nicht zwingend vorgeschrieben sind.

Eine wichtige Frage ist, inwieweit die Benutzungsschnittstelle dabei explizit beschrieben werden sollte, z.B. durch konkrete Bedienelemente wie Taster. Oft existieren Standardlösungen für diese Bedienelemente. Aber gerade durch die Ausweitung der Softwarefunktionalität ist bzgl. der Fahrerbedienung vieles im Umbruch (z.B. Verwendung von Displays anstelle von Tastern). Folgende Forderungen erscheinen deshalb sinnvoll:

(LH3) Im Lastenheft ist möglichst von Bedienelementen zu abstrahieren und nur auf die essentiellen Interaktionen zu fokussieren.

Wie in 2.1. diskutiert gilt gleiches auch für technische Elemente wie Sensoren und Aktuatoren:

(LH4) Im Lastenheft ist möglichst von Sensoren und Aktuatoren zu abstrahieren und nur auf die indirekten Größen zu fokussieren. Ist es nicht möglich lediglich indirekte Größen zu beschreiben, können natürlich auch direkte Größen beschrieben werden.

Bzgl. der Strukturierung fordern wir zur Unterstützung der Wiederverwendung:

(LH5) Die funktionalen Anforderungen im Lastenheft sollten nach Benutzeraufgaben strukturiert sein.

2.3 Umsetzung der Kriterien durch Richtlinien zur UC-Erstellung

UCs sind seit längerem zur Beschreibung der Benutzersicht auf das System etabliert [Ja95]. (LH1) und (LH5) können damit gut durch die Wahl von UC abgedeckt werden. UC werden durch UC Diagramme und durch textuelle Beschreibungen repräsentiert. Das UC Diagramm besteht aus “actors” (Akteuren) und UC (Anwendungsfällen) und deren Beziehungen. Die UC Diagramme listen die Aufgaben, die die Benutzer mithilfe des Systems erledigen können auf. Jeder UC im UC Diagramm wird in einer textuellen Beschreibung im Detail beschrieben. Die textuelle Beschreibung legt fest, wie die Benutzer mit dem Anwendungssystem interagieren, um ihre Aufgabe zu erledigen. Weiter werden Ausnahmesituationen bei der Benutzung des Systems und relevante Umgebungsvariablen, die im UC beobachtet bzw. kontrolliert werden müssen beschrieben. Abbildung 1 zeigt ein Beispiel für ein UC Diagramm. Abbildung 2 zeigt die textuelle Beschreibung eines UC aus dem Diagramm. Die Beschreibung von UC Diagrammen ist Bestandteil der UML [UML]. Zur ausführlichen textuellen Beschreibung sind einige Varianten bekannt, die sich aber in weiten Teilen überschneiden, vgl. z.B. [Co01].

Im Bereich der eingebetteten Systeme sind UCs noch weniger verbreitet, wenngleich sie in neueren Publikationen wie z.B. [HR02] bereits verwendet werden. Unklar ist allerdings, wie typische Elemente bei der Beschreibung eingebetteter Systeme (z.B. Sensoren und Aktuatoren) auf die Notation der UCs abgebildet werden. Unter Zugrundelegung des 4 Variablen Modells betrachten wir die konkreten Sensoren und Aktuatoren als Teil des Systems und nicht als Aktoren für UCs. Dies ist ein wesentlicher Unterschied zu anderen Anwendungen von UCs im Bereich eingebetteter Systeme, wie z.B. [HR02]. Werden Sensoren und Aktuatoren als Aktoren verwendet, so führt das nach unserer Erfahrung zu unnötig frühzeitiger Festlegung von technischen Details.

Im folgenden stellen wir am Beispiel der Funktionalität eines Türsteuergerätes unsere Richtlinien zur Erstellung von UCs vor und zeigen, wie die Anwendung der Richtlinien zur Erfüllung der in Kapitel 2.1 vorgestellten Kriterien beiträgt. Das Türsteuergerät ist realen

Steuergeräten bei DaimlerChrysler nachempfunden und in [HP02] dokumentiert. Wir betrachten im weiteren insbesondere die Funktionalität zur Einstellung des Fensters.

Abbildung 1 zeigt ein UC Diagramm für diese Fenstereinstellung. Abbildung 2 zeigt die textuelle Beschreibung der Funktionalität zur totalen Öffnung oder Schließung des Fensters, d.h. des UC „Steuere_Fensterscheibe_Total_an“.

Im folgenden geben wir einen groben Überblick über die Schritte, nach denen dieses UC Diagramm und die textuelle Beschreibung des UC „Steuere Fensterscheibe Total an“ entstanden sind.

In *Schritt 1* werden zunächst alle Aktoren ermittelt, die für das System (d.h. also die betrachtete Teilfunktionalität im Auto) relevant sind. Für die Ansteuerung der Fensterscheibe ergeben sich demnach drei verschiedene Aktoren (Fahrer, Beifahrer, Fond-Insasse), die direkt mit dem System interagieren können.

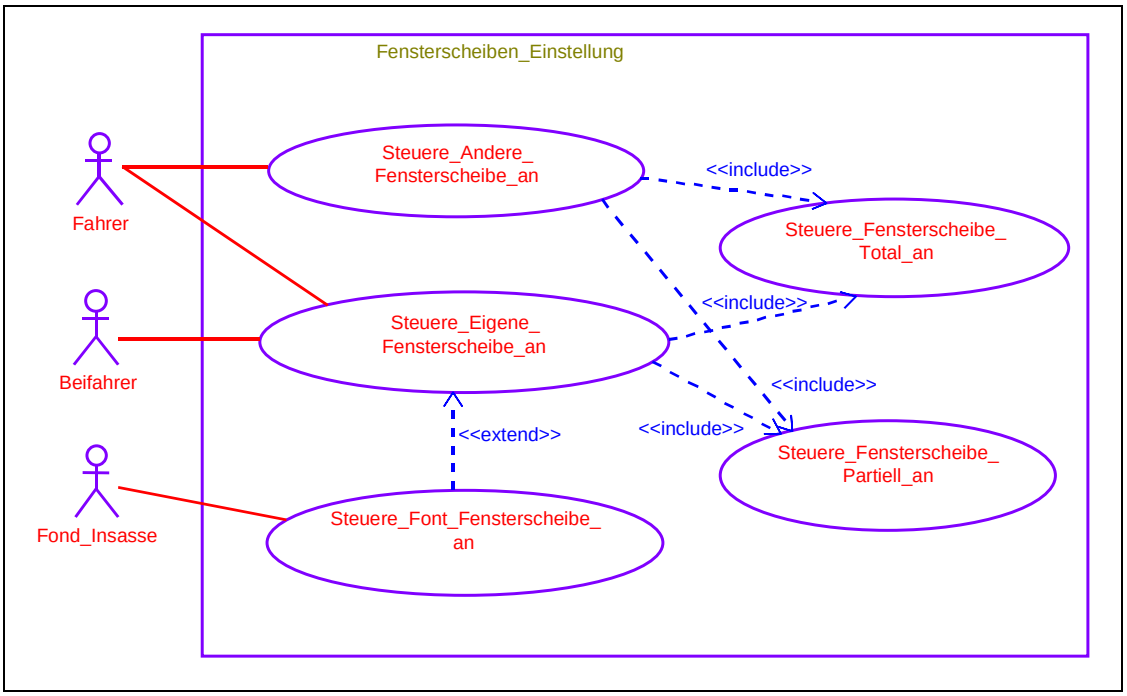

Abbildung 1: UC-Diagramm zur Fenstereinstellung

Jeder dieser Aktoren benutzt das System, um eine bestimmte Aufgabe zu erfüllen, wie z.B. die eigene Fensterscheibe oder eine andere Fensterscheibe im Fahrzeug einzustellen.

Deshalb wird in *Schritt 2* eine Liste der Aufgaben jedes Aktors erstellt und in einem UC Diagramm visualisiert. Jede Aufgabe entspricht dann einem UC. Zunächst könnte man sich einen allgemeinen UC „Steuere Fensterscheibe an“ vorstellen. In Abbildung 1 ist dieser UC

allerdings schon verfeinert in die UCs „Steuere_Andere_Fensterscheibe_an“, „Steuere_Eigene_Fensterscheibe_an“ und „Steuere_Fond_Fensterscheibe_an“. Die folgenden Schritte der Richtlinien leiten einen Anforderungs-Ingenieur an, die einzelnen Elemente des UC Templates auszufüllen (vgl. Abbildung 2) und zeigen wie die Verfeinerung des UC Diagramms durchgeführt wird.

In *Schritt 3* werden *Name* und *Aktor* direkt aus dem UC Diagramm übernommen. Danach wird das *Ziel*, das die Aktoren bei der Erfüllung der Aufgabe verfolgen, beschrieben. Die Vor- und Nachbedingungen beschreiben, welche Voraussetzungen gelten müssen, bevor der UC ausgeführt werden kann, bzw. welche Bedingungen gelten müssen, nachdem der UC erfolgreich beendet wurde.

In *Schritt 4* werden nun die monitorierten Größen im UC Template-Element *Eingänge* und die kontrollierten Größen im Template-Element *Ausgänge* beschrieben. Diese Größen repräsentieren die Schnittstelle zwischen dem System und der Umgebung. Monitorierte Größen sind Eingaben, mit denen ein bestimmtes Systemverhalten durch den Aktor angestoßen wird bzw. Umgebungsgrößen, die das Systemverhalten beeinflussen. Kontrollierte Größen sind Ausgaben, durch die das System die Systemumgebung beeinflusst. Bei der Festlegung der monitorierten und kontrollierten Größen ist entsprechend (LH3) darauf zu achten, dass von konkreten Bedienelementen abstrahiert wird, es sei denn, vom Auftraggeber ist dies explizit gewünscht. Wir sprechen deshalb in dem obigen UC nicht von „Fenstertastern“, mit denen der Fahrer sein Fenster ansteuert, sondern vom „Fahrer-Input“. Dadurch ist jede Art von Bedienelement in der weiteren Modellierung denkbar (Taster, Display, etc.). Ebenso sollten keine Sensor- und Aktuatornamen verwendet werden, wenn sie nicht als Realisierung zwingend sind. Dadurch kann insbesondere die Einhaltung des Kriteriums (LH4) sichergestellt werden.

Name	*Steuere Fensterscheibe total an*
Aktor	FahrerIn, BeifahrerIn, Fond_InsasseIn
Ziel	Aktor stellt die gewünschte Endposition der Fensterscheibe ein
Vorbedingungen	Keine
Beschreibung	1. Aktor gibt totale Ansteuerung und Richtung für Fensterscheibe vor 2. System bewegt Fensterscheibe in die angegebene Endposition [Ausnahme: Aktor steuert Fensterscheibe partiell an] [Ausnahme: Einklemmschutz wird aktiv] [Ausnahme: Limitposition erreicht] [Ausnahme: technisches Problem]
Ausnahmefälle	• Aktor steuert Fensterscheibe partiell an: -> Use Case *„Steuere Fensterscheibenposition partiell an“* • Einklemmschutz wird aktiv: sofortiges Herunterfahren der Fensterscheibe in Endposition, keine Unterbrechung möglich • Limitposition erreicht: System reagiert nicht • Technisches Problem: System reagiert nicht

Regeln	Das System muss den Einklemmschutz aktivieren, wenn bei Aufwärtsbewegung keine Fensterscheibenbewegung festgestellt wird
Qualitätsanforderungen	Die Bewegung der Fensterscheibe muss innerhalb von 3 Sekunden beendet sein
Eingänge	• Fensterscheibenbewegungsrichtung hoch/runter • Ist-partielle/totale Ansteuerung • Aktuelle Fensterscheibenbewegung
Ausgänge	Neue Fensterscheibenposition
Nachbedingungen	Fensterscheibe ist in Endposition

Abbildung 2: Textuelle UC-Beschreibung des UC „Steuere Fensterscheibe total an“

In *Schritt 5* werden im Element *Vorbedingungen* alle Bedingungen spezifiziert, die vor der Ausführung des UC gelten müssen. Im Template-Element *Beschreibung* werden, für die in Schritt 2 gesammelten Aufgaben, die Interaktion zwischen Aktor und System detailliert. Wir verwenden dabei die Ideen der essentiellen UC aus [CL01]. Bei der Beschreibung sind insbesondere Ausnahmefälle zu berücksichtigen, d.h. Situationen, in denen die Aufgabe nicht erfüllt werden kann. Die Reaktion auf bestimmte *Ausnahmefälle* wird in einem eigenen Template-Element behandelt. Unterscheidet sich das Systemverhalten in Abhängigkeit von der monitorierten Größe bzw. den Aktoren, von denen das Systemverhalten ausgelöst wird, dann sind die Aufgaben in Teilaufgaben zu unterteilen, und entsprechend das UC Diagramm zu verfeinern. Diese Verfeinerung führt zu der in Abbildung 1 beschriebenen Struktur des UC Diagramms. Solche Teilaufgaben sind im UC Diagramm als inkludierte UCs zu visualisieren. Weiter erkennt man in Abbildung 1, dass z.B. die Fensteransteuerung unterschiedlich ist, je nach Aktor, der Fensterscheibe ansteuert. Hintergrund ist, dass z.B. für den Beifahrer nicht relevant ist, ob die Kindersicherung aktiviert ist, aber ein Fond-Insasse kann die Fensterscheibe bei aktivierter Kindersicherung nicht bewegen. Diese Art der Verfeinerung führt dazu, dass die ursprüngliche denkbare Aufgabe „Steuere Fensterscheibe an“ aufgespalten wird in die Teilaufgaben „Steuere_Andere_Fensterscheibe_an“, „Steuere_Eigene_Fensterscheibe_an“ und „Steuere_Fond_Fensterscheibe_an“. In Schritt 5 ist insbesondere auch (LH2) zu beachten, d.h., dass nur für den Benutzer sichtbare Systemreaktionen beschrieben werden.

Im Template-Element *Regeln* sind Vorgaben festzuhalten, welche die Ausführung der Systemfunktionen näher festlegen, aber nicht wesentlich für die Interaktion sind. In diesem Beispiel ist detailliert, wie die indirekte Größe „Blockierung“ durch die direkte Größe „keine Aufwärtsbewegung“ abgebildet ist. Im Template-Element *Qualitätsanforderungen* sind qualitative Anforderungen an das Systemverhalten zu spezifizieren. Im Element *Nachbedingung* werden Bedingungen spezifiziert, die nach der Ausführung des UC gelten müssen.

3 Beschreibung von funktionalen Anforderungen im Pflichtenheft mit StateCharts

3.1 Anforderungen an das Pflichtenheft

Das Pflichtenheft dient dem Verantwortlichen für die Systementwicklung (in der Automobilentwicklung oft ein Zulieferer) dazu, das Systemverhalten so genau festzulegen, dass im Detail entschieden werden kann, ob und mit welchem Aufwand das Verhalten realisierbar ist und ob beispielsweise sehr innovative Lösungen gefragt sind. Weiter brauchen die Entwickler eine klare Vorgabe, was zu realisieren ist. Dies bedeutet insbesondere, dass das Zusammenspiel der Use Cases detailliert werden muss, um zu einer vollständigen Beschreibung des Systemverhaltens zu kommen. Wie im Lastenheft gilt aber, dass Systeminterna nur wichtig sind, wenn sie zwingend vom Auftraggeber vorgeschrieben sind. Um sicherzustellen, dass der Automobilhersteller das erstellte System abnimmt, muss das Pflichtenheft die Anforderungen aus dem Lastenheft umsetzen. Wie im Lastenheft ist eine gute Strukturierung zur Unterstützung von Änderbarkeit und Wiederverwendung wichtig. Zur Modellierung der funktionalen Anforderungen im Pflichtenheft werden Klassendiagramme und Statecharts verwendet.

3.2 Kriterien zur Verhaltensbeschreibung im Pflichtenheft

Aufbauend auf den im Abschnitt 3.1 diskutierten Überlegungen lassen sich Kriterien an die Beschreibung der Funktionalität im Pflichtenheft aufstellen. Diese Kriterien sind im Einzelnen:

> **(PH1)** Das Pflichtenheft repräsentiert eine vollständige Beschreibung des Systemverhaltens als Vorgabe für die Entwickler.
>
> **(PH2)** Es muss effizient statisch überprüfbar sein, ob das Pflichtenheft die Anforderungen aus dem Lastenheft umsetzt, d.h. Elemente aus dem Lastenheft müssen einfach auf Elemente im Pflichtenheft abgebildet sein.
>
> **(PH3)** Teilfunktionalität muss schnell lokalisierbar sein, um Wiederverwendung zu erleichtern.

Zur vollständigen Beschreibung des Systemverhaltens muss das Zusammenspiel der Use Cases detailliert werden. Dadurch wird aber die Beschreibung deutlich komplexer und die Überprüfung der internen Konsistenz deutlich schwieriger. Letzteres wird durch die Ausführbarkeit der Verhaltensbeschreibung unterstützt. Daher nehmen wir das folgende Kriterium hinzu:

> **(PH4)** Die Verhaltensbeschreibung muss ausführbar sein.

Oft ist eine ausführbare und vollständige Systembeschreibung mit der Entwurfsbeschreibung vermischt. Um bei der Entwicklung Aufwand zu sparen, kann es sinnvoll sein, statt einem Pflichtenheft eine Entwurfsbeschreibung zu verwenden. Dabei besteht jedoch die Gefahr, dass

zu früh und unreflektiert Gesichtspunkte wie Redundanzfreiheit oder Minimalität von Nachrichtenaustausch von Komponenten eingeführt werden, und damit die Freiheit der Entwickler unnötig eingeschränkt wird. Dies ist insbesondere bei Änderungen und Wiederverwendung gefährlich, weil dann nicht mehr klar trennbar ist, ob eine Anforderung an das Systemverhalten aus Auftraggebersicht notwendig ist, oder von Entwurfsüberlegungen beeinflusst ist. Wir fordern deshalb im folgenden,

> **(PH5)** dass Entwurfsüberlegungen nicht ins Pflichtenheft eingehen, wenn sie nicht zwingend vom Auftraggeber vorgeschrieben sind.

Wir wollen damit insbesondere demonstrieren, dass eine ausführbare und vollständige Systembeschreibung auf Anforderungsebene möglich ist. Aus dem gleichen Grund fordern wir auch auf dieser Ebene,

> **(PH6)** so weit wie möglich von konkreten Bedienelementen oder internen technischen Problemen (wie z.B. Ausfall eines Sensors) zu abstrahieren.

Aufgrund der hohen Komplexität der Systemfunktionalität ist auf dieser Ebene die Verständlichkeit und Einfachheit der Verhaltensbeschreibung eines der wichtigsten Qualitätskriterien. Aus diesem Grund fordern wir weiter,

> **(PH7)** die Komplexität der Verhaltensbeschreibung so gering wie möglich zu halten.

3.3 Umsetzung der Kriterien durch Richtlinien zur SC-Erstellung

Wie in [Ki95] beschrieben, bieten sich in der Automatisierungstechnik zur Erfüllung dieser Kriterien Zustands(übergangs)diagramme als Notation an. Zusätzlich verwenden wir in unserem Ansatz Klassendiagramme.

Ein *Klassendiagramm* beschreibt Klassen, Attribute und Methoden von Klassen und Beziehungen zwischen Klassen innerhalb eines Teilsystems und zu anderen Teilsystemen. Es beschreibt somit die statische Struktur eines bestimmten Teilsystems. *Zustandsdiagramme* beschreiben das dynamische Verhalten einzelner Klassen. Für jede Klasse mit dynamischen Verhalten wird ein Zustandsdiagramm erstellt. Eine Klasse hat ein dynamisches Verhalten, wenn sie sich durch verschiedene Zustände auszeichnet und das Verhalten der Klasse abhängig von ihren verschiedenen Zuständen ist. Ein Zustandsdiagramm beschreibt verschiedene Zustände und Zustandsübergänge zwischen den Zuständen für die Objekte einer Klasse. Nicht berücksichtigt werden im Zustandsdiagramm die Aufrufe von Methoden der zugehörigen Klasse, die lediglich zum Setzen oder Abfragen von Werten dienen und keine Zustandsänderung auslösen.

Die Bedeutung von Zustandsübergangsdiagramme in der Automatisierungstechnik wird insbesondere in der Automobilindustrie durch die Verwendung von Modellierungswerkzeuge wie STATEMATE oder MATHLAB/SIMULINK deutlich.

Es gibt viele Varianten von Zustandsdiagrammen. Aufgrund der vorhandenen Werkzeugunterstützung sind insbesondere Harels Statecharts weit verbreitet [HP98]. In [Gl02] werden die wichtigsten Elemente verschiedener SC Varianten diskutiert und semantische und syntaktische Verbesserungen vorgeschlagen. Wir verwenden im folgenden die durch das Tool Rhapsody unterstützte Syntax und Semantik. Die Grundlagen dieses Werkzeugs sind in [HG97] beschrieben. Ein wesentlicher Vorteil dieses Werkzeugs gegenüber STATEMATE ist, dass die Strukturierung auch durch objektorientierte Konstrukte wie Klassen unterstützt wird. Dies ermöglicht insbesondere die Beschreibung gleichzeitigen und gleichartigen Verhaltens durch Instanzen einer Klasse und vereinfacht dadurch Wiederverwendung und verbessert die Verständlichkeit der SCs. Die Wahl dieses Werkzeugs unterstützt somit die Kriterien (PH4) und (PH1), sowie aufgrund der OO-Konstrukte auch (PH3). Für die Verwendung von objektorientierten Elementen berücksichtigen wir insbesondere auch die Überlegungen aus [vKP99].

Im Folgenden werden wir wieder am Beispiel unsere Richtlinien zur Erstellung von SCs vorstellen und zeigen, wie die Anwendung dieser Richtlinien zur Erfüllung der Kriterien aus Kapitel 3.2 beiträgt. Die Abbildungen 3 – 8 zeigen Teile der Umsetzung des Use Cases aus Abb. 2, die unter Anwendung der Richtlinien zur Erstellung von SCs entstanden sind.

Alle Schritte der Richtlinien leiten die Entwickler der SCs dazu an, eine vollständige Beschreibung des Systemverhaltens zu erstellen. Daher wird das Kriterium (PH1) durch den schrittweisen Übergang von den UCs zu den SCs realisiert.

Für die Beschreibung aller SCs gilt im weiteren, dass die Namen der Zustände und die Namen von Ereignissen, Operationen und Attributen so gewählt werden sollten, dass diese intuitiv für den Leser verständlich sind. Diese Vorgabe trägt insbesondere zur Realisierung von (PH7) bei.

(Schritt 1): Erstellung eines Klassendiagramms

In diesem Schritt wird zunächst, ausgehend von der UC-Beschreibung die Klassenstruktur des Systems erzeugt. Die Richtlinien leiten den Entwickler systematisch dazu an, die Klassenstruktur zu erstellen, d.h. sie geben vor, welche Informationen aus den UCs sich in welchen Elementen der SCs widerspiegeln. Zunächst werden alle UCs des Lastenhefts, die *eigene Funktionalität* beschreiben, als Klasse modelliert. Ein UC beschreibt *keine eigene Funktionalität*, wenn er ausschließlich andere UCs inkludiert, also nur der Strukturierung dient. Dies reduziert die Größe der SCs und dient damit Kriterium (PH7).

Abbildung 3 zeigt einen Teil des Klassendiagramms der Fallstudie. Jeder in Abbildung 1 dargestellte UC wird hier als Klasse repräsentiert. Die Beziehungen zwischen den UCs, die im UC-Diagramm sichtbar sind, werden auf Assoziationen zwischen den Klassen abgebildet. Die Includes-Beziehungen werden als bidirektionale Assoziation modelliert und die Extends-Beziehung als Vererbungsbeziehung.

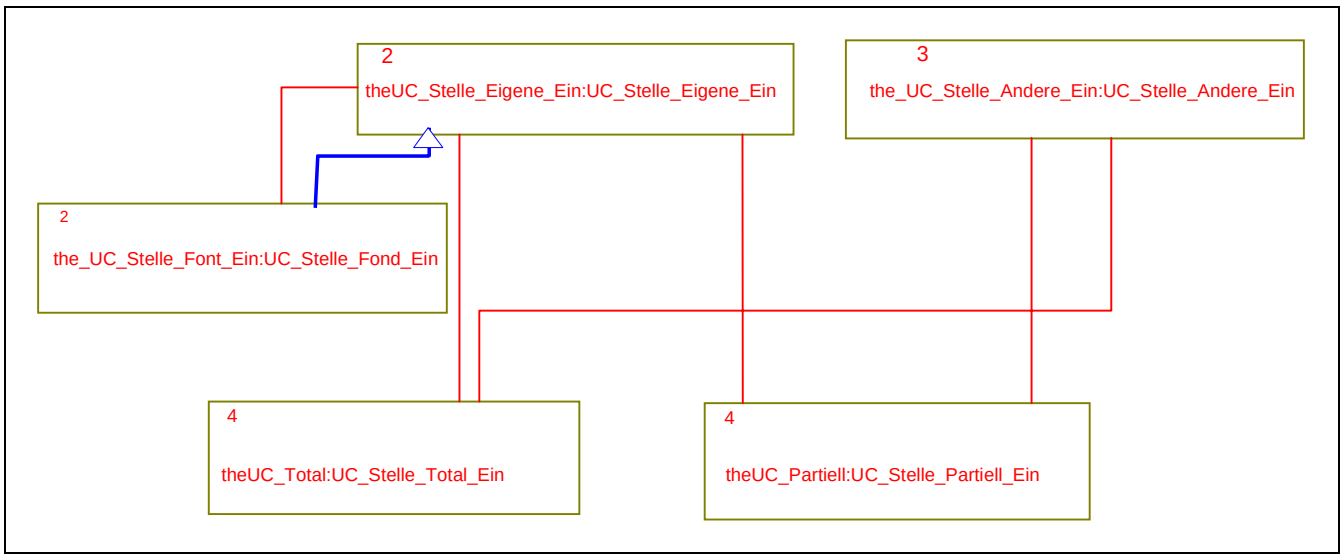

Abbildung 3: UC-Klassen

Weiter werden alle monitorierten und kontrollierten Größen aus der UC-Beschreibung als Klasse modelliert. In diesem Teilschritt ist allerdings zu überprüfen, ob es möglich ist, einzelne monitorierte oder kontrollierte Größen zu komplexeren Größen zusammenzufassen. Insbesondere Größen, die den gleichen Wertebereich besitzen, können ggf. zusammengefasst werden. Auch diese Vorgabe unterstützt Kriterium (PH7), da so nur für das Systemverhalten relevante Aspekte der Größen modelliert werden. Abbildung 4 zeigt die Klassen der monitorierten Variablen des in Abbildung 2 beschriebenen UCs zur Fenstereinstellung. Die in diesem UC beschriebenen Eingänge („Fensterscheibenbewegungsrichtung-hoch-runter“ und „Ist-partielle/total Ansteuerung“) werden, entsprechend der verschiedenen Aktoren des UC, durch drei Klassen (M_Fahrer_Input, M_Beifahrer_Input, M_Fond_Insassen_Input) repräsentiert, die diese Eingänge kapseln. Die Kindersicherung ist eine Eingangsgröße für die Fenstereinstellung allgemein und wird deshalb ebenfalls als Klasse repräsentiert. Die Assoziationen deuten Verbindungen zu den Klassen an, welche die UCs repräsentieren, die diese Größen als Input besitzen.

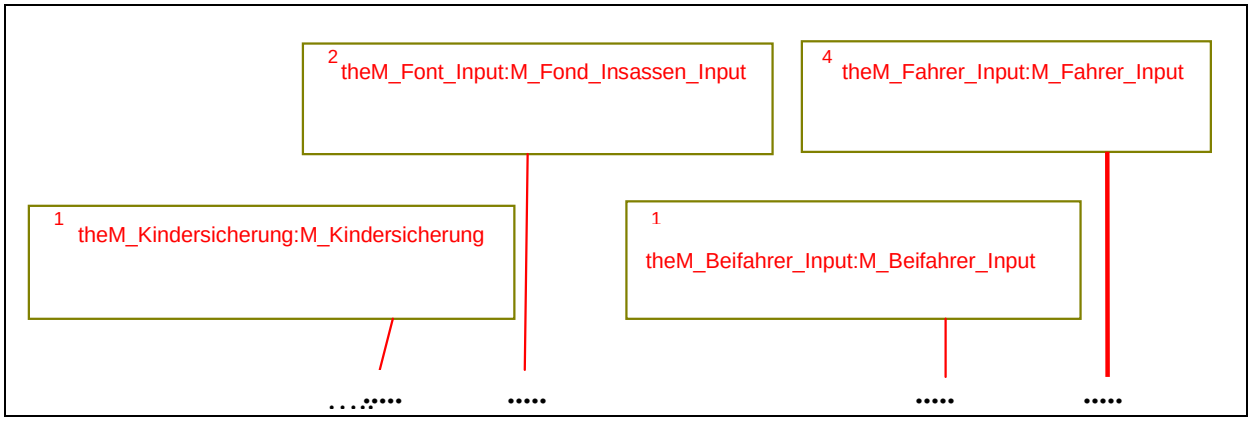

Abbildung 4: Ausschnitt aus dem Klassendiagramm (monitorierten Variablen)

Wir haben die Modellierung der UC-Klassen in der jetzigen Form gewählt, da wir durch diese Art der Abbildung die Realisierung der Kriterien (PH1), (PH2), (PH3), und (PH7) unterstützen können. Insbesondere durch die Abbildung der UCs auf Klassen können im Lastenheft beschriebene Teilfunktionalitäten sehr schnell im Pflichtenheft lokalisiert werden (PH3). Weiter wird durch diese explizite Abbildung der Elemente aus dem UC-Template die Verfolgbarkeit zwischen Lasten- und Pflichtenheft und dadurch die Überprüfung der Umsetzung der im Lastenheft dargestellten Funktionalität erleichtert (PH2). Durch die Bildung von Instanzen der verschiedenen UC-Klassen ist die Modellierung gleichzeitig aktiver UCs möglich. Zum Beispiel ist es möglich, dass der Fahrer und der Beifahrer gleichzeitig ihr Fenster einstellen. In diesem Fall wäre der UC „Stelle_Eigene_Fensterscheibe_Ein“ zweimal aktiv. Dadurch wird insbesondere (PH1) sichergestellt. Ein weiterer Vorteil der Abbildung auf Klassen ist die damit verbundene Möglichkeit das Vererbungskonzept zu nutzen, d.h. Funktionalitäten die mehreren UCs gemein sind in einer Superklasse zu kapseln, was die Erfüllung von (PH3) und (PH7) unterstützt.

Allerdings bringt diese Art der Modellierung auch Nachteile mit sich. Durch die explizite Modellierung der Instanzen der UC-Klassen ist es nötig, in den SCs explizit die Kommunikation zwischen den Instanzen zu modellieren. Diese Art der expliziten Modellierung der Kommunikation zwischen den Instanzen entspricht dem Ansatz von Glintz, der ebenfalls empfiehlt, anstelle eines globalen Broadcast-Konzeptes Kommunikation zwischen einzelnen Objekten zu modellieren [GI02]. Danach sollte das Broadcast-Konzept nur lokal in einzelnen Objekten verwendet werden, um eine globale Kopplung der Objekte zu vermeiden.

(Schritt 2 & Schritt 3): Modellierung der SCs der monitorierten und kontrollierten Größen

Durch die Modellierung von SCs für die Klassen der monitorierten und kontrollierten Größen ist es möglich, in den Zuständen dieser SCs Ereignisse zu puffern, die für das weitere Systemverhalten von Bedeutung sein können und daher nicht verloren gehen dürfen . Daher muss sich der Entwickler zur Vorbereitung dieses Schrittes Gedanken machen, welche Werte eine monitorierte oder kontrollierte Größe annehmen kann. Dabei muss er oder sie insbesondere darauf achten, dass mögliche Realisierungen der Bedienelemente die Modellierung der Größen nicht beeinflussen (PH6).

Aufbauend auf diesen Vorüberlegungen wird der Entwickler in den Schritten 2 und 3 angeleitet, die Beschreibung der monitorierten und kontrollierten Größen zu verfeinern. Auch dieses Vorgehen, wurde auf ähnliche Weise in [GI02] beschrieben, wo SC als Verfeinerung von Objektbeschreibungen auf unterschiedlichen Hierarchieebenen empfohlen werden. Unsere Richtlinien geben darüber hinaus Hinweise, wie der Übergang zwischen diesen Hierarchieebenen erreicht werden kann. Dazu werden zu allen Klassen, die monitorierte oder kontrollierte Größen beschreiben, SCs erstellt. Eine Ausnahme bilden Klassen, die eine Größe beschreiben, deren mögliche Werte nicht durch Äquivalenzklassenbildung sinnvoll auf einen diskreten Wertebereich abgebildet werden können. Die Werte solcher Größen werden als

Attribute der entsprechenden Klasse modelliert. Für alle anderen Größen wird ein SC erstellt, das als Zustände die Werteklassen enthält, welche die modellierte Größe annehmen kann (vgl. auch Abbildung 5).

Die Transitionen repräsentieren Ereignisse, die in der Umgebung des Systems auftreten, d.h. die von den Aktoren des Systems ausgelöst werden. Solche Ereignisse sind in dem UC-Template-Element *Beschreibung* genannt. Häufig geben Sätze, in denen der Aktor der UCs das Subjekt des Satzes ist, Hinweise auf solche Ereignisse.

Die Abbildung 5 beschreibt das SC der monitorierten Größe „Kindersicherung“, die eine Eingangsgröße des UC „Stelle_Fond_Fensterscheibe_Ein“ ist. In den Zuständen der Variable können die Ereignisse „ev_Aktiviere_Ksicherung“ und „ev_Deaktiviere_Ksicherung“ gepuffert werden.

Das SC dieser monitorierten Größe verdeutlicht einen weiteren sehr wichtigen Aspekt bei der Modellierung von Transitionen. Bei monitorierten Größen, deren Zustände dazu dienen, Ereignisse aus der Umgebung zu puffern, erfolgt keine aktive Weiterleitung der Ereignisse aus der Umgebung an das System. Dies ist in Abbildung 5 daran zu erkennen, dass die Transitionen lediglich durch ein Ereignis ausgelöst werden, aber keine weiteren Aktivitäten definiert sind, die beim Auslösen der Transition abgearbeitet würden. Bei monitorierten Größen, die Ereignisse aus der Umgebung beobachten und diese direkt an das System weiterleiten, müssen an den Transitionen entsprechend Aktivitäten definiert werden, die diese Weiterleitung realisieren.

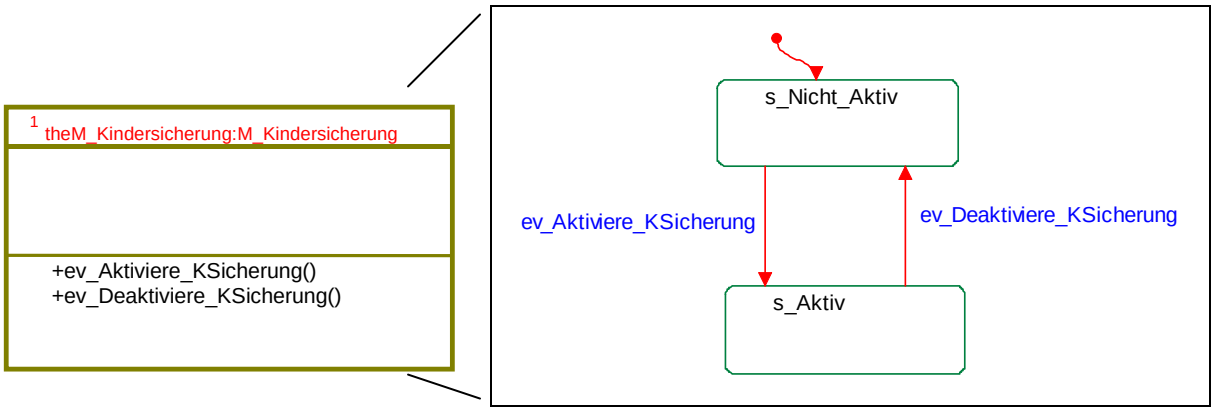

Abbildung 5: Klasse und SC der Größe Kindersicherung

(Schritt 4): Modellierung der SCs der UC-Klassen

In Schritt 4 wird nun die Beschreibung der Klassen verfeinert, die UCs repräsentieren. Dazu wird zu jeder UC-Klasse ein SC erstellt, das initial zwei Zustände enthält. Einen „Idle-Zustand“, der eingenommen wird, wenn der entsprechende UC gerade nicht aktiv ist und einen „System-Reaction-Zustand“, der eingenommen wird, wenn der UC gerade ausgeführt wird. Diese einfache Struktur erleichtert die Verfolgbarkeit zu den UCs und somit die Einhaltung von (PH2), (PH3) und (PH7).

Die Übergänge zwischen diesen Zuständen werden von mehreren Elementen der UC-Beschreibung (Regeln, Ausnahmefälle, Beschreibung) beeinflusst. Zunächst sind die Transitionen zu modellieren, die das Starten bzw. das Beenden des UC beschreiben. Das Starten des UC entspricht dem ersten Schritt der UC-Beschreibung und wird durch eine Transition vom Idle-Zustand in den System-Reaction_Zustand repräsentiert. Ein UC ist dann beendet, wenn alle Schritte des Szenarios durchlaufen sind und ggf. die Nachbedingung erfüllt ist. Dies wird modelliert durch eine Transition vom System-Reaction-Zustand in den Idle-Zustand. Die Ereignisse, die diese Transitionen auslösen, ergeben sich aus dem UC-Template-Element *Beschreibung*.

Abbildung 6 zeigt das SC der Klasse, die den UC „Steuere Fensterscheibe total an“ aus Abbildung 2 umsetzt, nachdem die Zustände und die Transitionen zum Starten und Beenden des UC eingefügt wurden. Das Ereignis „ev_Start_Total“ wird von der monitorierten Größe erzeugt, welche die Input-Möglichkeiten der Aktoren modelliert. Dadurch wird ein Ereignis aus der Umgebung an das System weitergeleitet. Die Transition von „s_Idle“ nach „s_System_Reaction“ entspricht der ersten Systemreaktion, nachdem der Aktor eine Eingabe in das System gemacht hat.

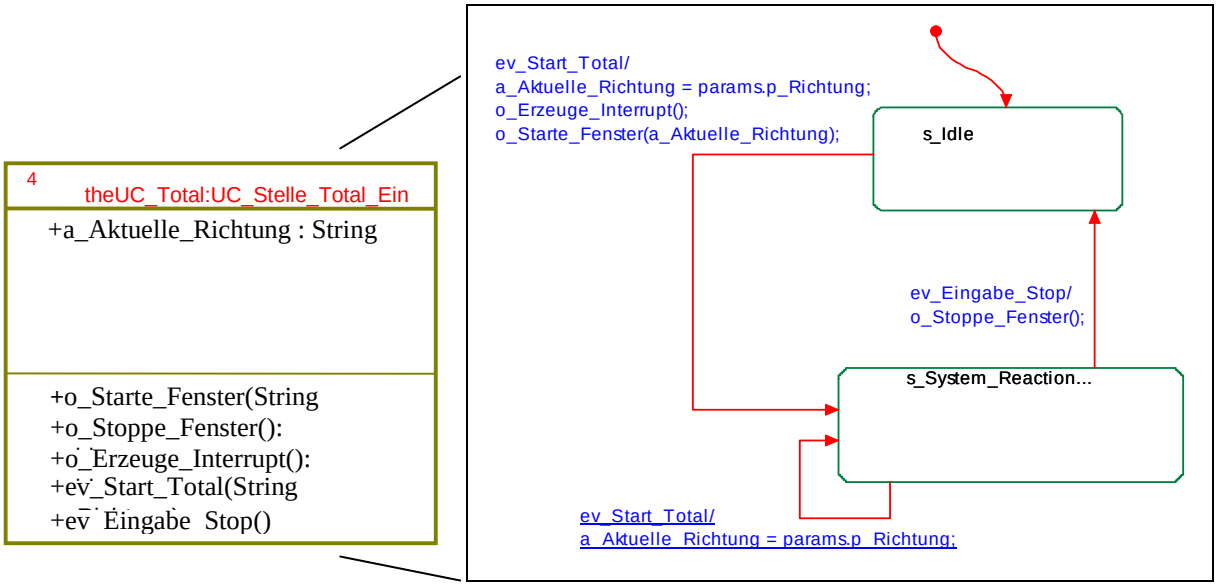

Abbildung 6: Klasse und SC des UC "Steuere Fensterscheibe total an"

Die an den Transitionen beschriebenen Operationen „o_Starte_Fenster()“ und „o_Stoppe_Fenster()“ dienen dazu, das Starten bzw. Beenden des UC an die entsprechenden Fenster zu propagieren. Diese Informationen können aus dem UC-Template-Element *Beschreibung* gewonnen werden. Die Operation „o_Erzeuge_Interrupt()“ dient dazu, einen der im UC beschriebenen Ausnahmefälle zu realisieren. Die Ausführung eines anderen UC auf dem gleichen Fenster wird durch den vorliegenden UC abgebrochen. Deshalb muss ein entsprechendes Ereignis im UC „Steuere Fensterscheibe_Total_ an“ erzeugt werden, was über die Operation „o_Erzeuge_Interrupt()“ erreicht wird (vgl. nächsten Absatz). Die zyklische

Transition im System-Reaction-Zustand ist notwendig, um sicherzustellen, dass ein Wechsel der Bewegungsrichtung auch während der Ansteuerung des Fensters vom System wahrgenommen wird.

Die Zustandsübergänge werden auch von den in den UCs beschriebenen Ausnahmefällen beeinflusst. Wir unterscheiden zwischen drei Arten von Ausnahmen: Ausnahmen, die den Start der System-Reaktion verhindern; Ausnahmen, die eine System-Reaktion abbrechen; Ausnahmen, die aus der Unterbrechung durch andere UCs resultieren. Informationen über die zu berücksichtigenden Ausnahmefälle geben die UC-Template-Elemente *Beschreibung* und *Ausnahmefälle*. Das erstellte SC-Modell eines UC ist zu erweitern, um diese Ausnahmen zu berücksichtigen. Dazu sind Ausnahmen, die eine System-Reaktion verhindern, als Guard an der Transition zu modellieren, die vom Idle-Zustand in den System-Reaction-Zustand führt. Ausnahmen, die einen Abbruch der System-Reaktion verursachen und Ausnahmen, die durch Unterbrechung von anderen UCs entstehen, werden als zusätzliche Transitionen vom System-Reaction-Zustand in den Idle-Zustand modelliert. Das Ereignis, das diese Transitionen auslöst, muss dann so benannt werden, dass offensichtlich ist, welche Ausnahme aus dem UC durch das Ereignis realisiert wurde.

Abbildung 7 zeigt erneut das SC der Klasse, die den UC „Steuere Fensterscheibe Total an“ repräsentiert. In diesem SC sind nun zwei weitere Transitionen hinzugekommen. Die Transition, die durch das Ereignis „ev_Interrupt_Partiell“ ausgelöst wird, realisiert die Ausnahme, dass der beschriebene UC von einem anderen UC („Steuere Fensterscheibe partiell an“) unterbrochen wird. Die zweite zusätzliche Transition ist nötig, um die Ausnahme zu modellieren, dass das Fenster seine maximale bzw. minimale Aussteuerung erreicht hat. Die initiale Transition vom Zustand „s_Idle“ nach „s_System_Reaction“ wurde um einen Guard erweitert. Dieser Guard modelliert die im UC beschriebene Ausnahme, dass die System-Reaktion nicht gestartet werden kann, falls das Fenster bereits in seiner obersten oder untersten Position ist und eine Ansteuerung nach oben oder unten erfolgt.

Die in diesem Schritt beschriebene Realisierung der Ausnahmefälle stellt erneut die Verfolgbarkeit von bestimmten Elementen der UCs zu Elementen der SCs her und trägt somit zur Realisierung der Kriterien (PH2) und (PH3) bei.

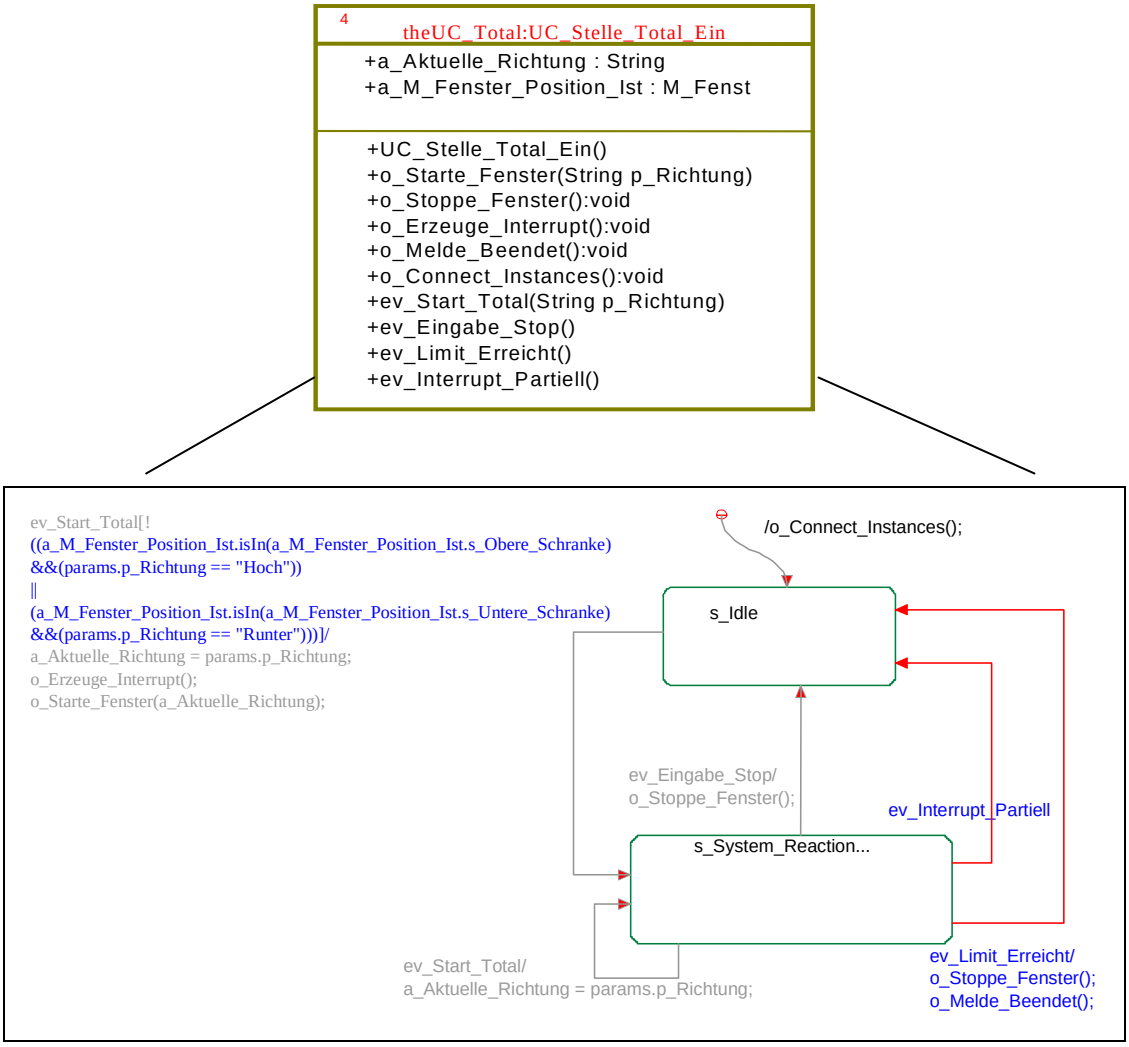

Abbildung 7: Erweitertes SC des UC "Steuere Fensterscheibe total an"

(Schritt 5): Umsetzung von Regeln/Verfeinern der System-Reaktion

Werden in den UCs Regeln beschrieben, die detailliertere Informationen zum Systemverhalten beschreiben, so müssen diese Regeln ebenfalls in den SCs abgebildet werden. Dazu muss eine Verfeinerung des System-Reaction-Zustandes erfolgen. Dadurch wird erreicht, dass alle Aspekte des UC, die das explizite Verhalten des UC betreffen, im System-Reaction-Zustand gekapselt sind (PH2), (PH3). Es sind allerdings zwei Fälle zu unterscheiden.Wenn die Verfeinerung des System-Reaction-Zustandes visualisierbar sein sollte, dann ist die Verfeinerung in einem Sub-Statechart zu realisieren. In diesem Fall kann die korrekte Realisierung durch eine Simulation des ausführbaren Modells sichtbar gemacht werden. Wenn die Visualisierbarkeit der Verfeinerung nicht notwendig ist, dann sollte die Regel als Operation modelliert werden, die im System-Reaktion-Zustand aufgerufen wird. Durch diese Art der Modellierung wird die Komplexität der SCs niedrig gehalten (PH7).

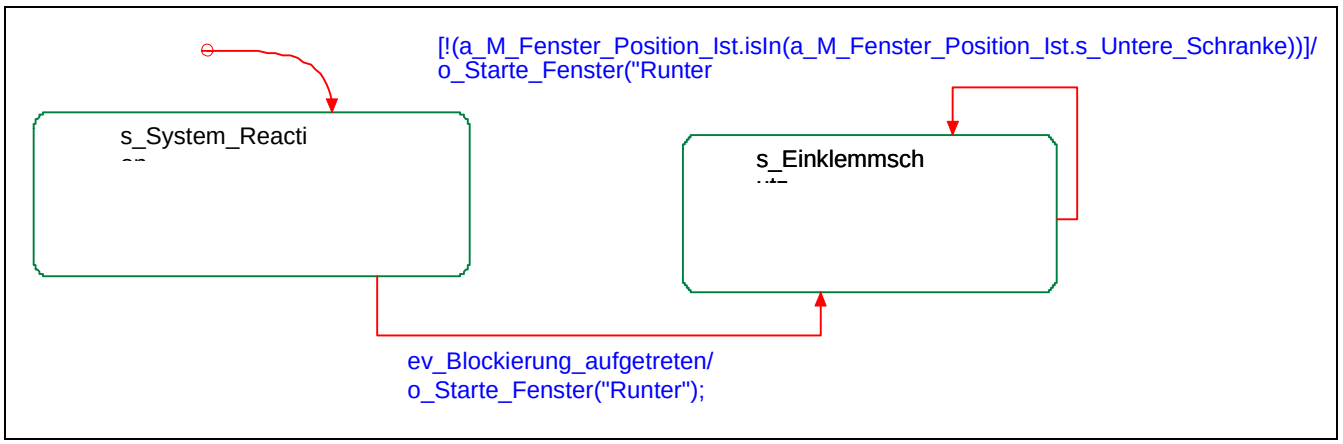

Abbildung 8: Verfeinerung des Zustandes s_System_Reaction

Abbildung 8 zeigt die Verfeinerung des Zustandes „s_System_Reaction“ durch ein Sub-Statechart dieses Zustandes. In diesem Sub- Statechart wird die im UC „Steuere Fensterscheibe total an“ beschriebene Regel realisiert, dass in bestimmten Situationen der Einklemmschutz zu aktivieren ist. Da diese Funktionalität sicherheitsrelevante Aspekte beschreibt, sollte die korrekte Realisierung der Funktionalität in dem ausführbaren Modell überprüfbar sein. Aus diesem Grund wurde die Verfeinerung der System-Reaktion mit Hilfe von Sub- Statecharts und nicht durch eine Operation realisiert.

3.4 Diskussion der SC Richtlinien

Es sind natürlich auch andere Vorgehen zur Erstellung von SC möglich und bekannt, allerdings nach Wissen der Autoren keine so detaillierten Richtlinien wie die hier vorgestellten. Auch in [HR02] werden SCs zur Präzisierung des Systemverhaltens verwendet, allerdings nur als optionale Ergänzung zu UCs und es wird auf Ausführbarkeit keinen Wert gelegt. Darüber hinaus sind die dort beschriebenen Richtlinien zur Ableitung von SCs aus UCs sehr grob. Nach unserem Verständnis sind unsere Richtlinien eine Konkretisierung des dort beschriebenen Vorgehens.

In der Praxis werden SCs oft erst auf Ebene des Entwurfs eingesetzt, in dem die SCs insbesondere auch bzgl. Redundanz optimiert sind. So könnte man in unserem Beispiel die SCs ähnlicher UCs wieder zusammenfassen und die unterschiedliche Systemreaktion z.B. durch Parameter modellieren. Dieser Schritt ist natürlich unbedingt notwendig zur Vorbereitung einer effizienten Codeerzeugung aus den SCs. Auf der Anforderungsebene ist aber die Verständlichkeit sehr viel wichtiger und daher spielen Optimierungen der SCs auf dieser Ebene eine eher untergeordnete Rolle. Darüber hinaus scheint es möglich, solche Optimierung dann (halb-) automatisch auf den von uns erzeugten SCs durchzuführen.

Es ist sehr wichtig, bereits auf der Ebene des Pflichtenheftes eine hohe Qualität der SCs sicher zu stellen, um zu verhindern, dass Fehler in den Entwurf übertragen werden. Neben anderen Techniken, wie zum Beispiel Inspektionen von SCs [De03], können wir in unserem Ansatz

mehrere Qualitätskriterien durch die Ausführbarkeit der SCs sicher stellen. Die Ausführbarkeit gewährleistet insbesondere eine Überprüfung der korrekten Interaktion zwischen verschiedenen SCs und stellt die Konsistenz zwischen den SCs und den Klassendiagrammen sowie zwischen verschiedenen SCs sicher. Die korrekte Interaktion verschiedener SCs untereinander kann in unserem Ansatz mit Hilfe von Simulationen der SCs im Tool Rhapsody analysiert werden. Dadurch können Fehler in der dynamischen Interaktion der SCs frühzeitig entdeckt und kostengünstig korrigiert werden.

4 Diskussion

Im folgenden fassen wir die wesentlichen Synergien und Unterschiede von UCs und SCs zusammen und diskutieren unsere bisherigen Erfahrungen mit unserem Vorgehen, vergleichbare Ansätze sowie weitere Arbeiten aus dem QUASAR Projekt.

4.1 Zusammenfassung und Erfahrungen

UCs ermöglichen eine vor allem textuelle Beschreibung der Benutzersicht auf ein System, die das Systemverhalten nicht vollständig erfasst. SCs ermöglichen eine graphische, ausführbare und vollständige, aber dennoch abstrakte, d.h. von Entwurfsdetails freie, Beschreibung. Beide Sichten sind unverzichtbar [WW02]. Die Herausforderung ist deshalb, die möglichst effiziente Erstellung beider Beschreibung zu unterstützen. Insbesondere bei der Erstellung von SCs ist ein Übergang von einer textuellen, unvollständigen und auf Benutzeraufgaben fokussierenden Beschreibung zu einer graphischen, vollständigen und auf Zustände fokussierenden Beschreibung zu leisten. Dies wird oft als ein Gegensatz empfunden, so dass die Erstellung von UCs als ein Umweg erscheint. Unsere Richtlinien zeigen, dass eine UC-Struktur eine hervorragende Ausgangsbasis für die Erstellung von SCs ist.

Um diese Aussage zu validieren, haben wir unseren Ansatz mit Studenten erprobt. In einem Seminar im SS02 an der Universität Kaiserslautern haben 4 Studenten mit einer Vorgängerversion unserer Richtlinien 2 Rhapsody-Modelle der Fenstereinstellung erstellt. Ausgangsbasis war für sie ein nach den Richtlinien erstelltes Modell der Sitzeinstellung in Rhapsody, die Use Cases für die Fenstereinstellung sowie die Richtlinien. Um die Nützlichkeit der Richtlinien zu validieren, sollten die Studenten während der Erstellung der SCs ihre Erfahrungen zu den einzelnen Schritten der Richtlinien dokumentieren. Alle Studenten waren in der Lage, mit Hilfe der Richtlinien ein ausführbares SC-Modell zu erstellen. Bei nahezu allen Schritten der Richtlinien berichteten die Studenten, dass die Anwendung des Schrittes ohne größere Probleme möglich war. Insbesondere die durch die Richtlinien geschaffene Verfolgbarkeit zwischen UCs und SCs wurde als sehr positiv bewertet. Aufgrund der Erfahrungen aus dem Seminar haben wir die Richtlinien noch weiter verbessert. Zur Zeit werden die überarbeiteten Richtlinien bei unserem Projektpartner am Fh FIRST erprobt.

4.2 Verwandte Arbeiten

Viele Arbeiten beschäftigen sich mit der (halb-)automatischen Ableitung von Szenarien aus SCs bzw. der Ableitung von SCs aus Szenarien, wobei unter Szenarien UCs, textuelle Szenarien und Message Sequence Charts (MSC), subsumiert sind. Diese Ansätze verwenden die Ableitung aber vor allem für die Validierung. Die Verständlichkeit, der erzeugten SCs, ist deshalb oft nicht das entscheidende Kriterium. In [So02] wird ein Algorithmus beschrieben, der aus einem UC ein SC erzeugt, das als Transitionen nur die Ein- und Ausgabeaktionen des Systems erhält. Um Schleifen erkennen zu können und auch SCs verschiedener UCs integrieren zu können, werden Zustände durch Prädikate charakterisiert. Dieser Algorithmus kann aber keine hierarchischen SCs erzeugen. Darüber hinaus ist das Systemverhalten nicht vollständig beschrieben, da z.B. mehrere Instanzen eines UC nicht berücksichtigt werden. Ein halbautomatischer Ansatz zur Erzeugung von hierarchischen SCs wird in [WS00] vorgestellt. Ausgangspunkt sind hier allerdings Message Sequence Charts, d.h. im Gegensatz zu UCs singuläre Abläufe. Aus diesen werden automatisch SCs erzeugt und schrittweise integriert. Dabei kann auch die für SCs typische Hierarchie erzeugt werden. Die entstehenden SCs müssen dann allerdings noch vervollständigt werden um Informationen, die in den MSCs nicht vorhanden sind. Ähnliches gilt für den in [RG99] vorgestellten SCENT-Ansatz. Hier wird halbautomatisch aus einzelnen UCs je ein SC erzeugt. Die Zustände der SCs entsprechen dabei den Aktionen von Benutzer oder System. Dabei wird insbesondere überprüft, ob die UC-Beschreibung in sich vollständig ist. Darauf aufbauend können Testfälle aus einzelnen SCs bzw. auch für das Zusammenspiel der SCs abgeleitet werden. Auch dieser Ansatz ist damit eher geeignet zur Validierung der UCs, um sicherzustellen, dass alle möglichen Interaktionen berücksichtigt wurden. Die resultierenden SCs beschreiben nicht das Systemverhalten vollständig, sondern das Interaktionsverhalten.

Auch bei der Erzeugung von Szenarien aus SCs wird typischerweise nicht Benutzersicht und Entwicklersicht voll unterstützt. In [Be02] wird ein Ansatz vorgestellt, in dem aufbauend auf das UC Diagramm inkrementell ein SC für das Systemverhalten erstellt wird. Dabei wird schrittweise Funktionalität für jeden UC hinzugenommen. Auf diese Weise ist aber die für den Benutzer wichtige Interaktion zwischen System und Benutzer nie explizit beschrieben. Darüber hinaus erscheint es zweifelhaft, dass das komplexe Verhalten eingebetteter Systeme ohne weitere Beschreibung aus einem UC-Diagramm abgeleitet werden kann.

4.3 Weitere Arbeiten

Der von uns beschriebene Übergang von UCs zu SCs kann sicherlich noch weiter optimiert werden. Insbesondere wollen wir die Richtlinien für die SCs bzgl. der Umsetzung von Regeln erweitern. Ein weiterer interessanter Punkt ist eine Erweiterung von Rhapsody, so dass ein Skelett der SCs automatisch aus den UCs erstellt wird.

Ebenso wollen wir in weiteren Arbeiten die Richtlinien für die UCs detaillieren und dabei insbesondere für eingebettete Systeme typische Muster für die Verwendung von natürlicher Sprache berücksichtigen [De02].

Danksagung

Diese Projekt wurde gefördert vom BMBF unter Kennzeichen VFG0004A. Wir danken unseren Projektkollegen am Fh FIRST, den Studenten des Seminars sowie Experten bei DaimlerChrysler für viele interessante Diskussionen zum Thema Anforderungsbeschreibung mit SCs und UCs.

5 Literatur

[Be02] H. Behrens: Requirements Analysis and Protytping using Scenarios and Statecharts, ICSE-Workshop on Scenarios and State Machine Models, Algorithms and Tools, Orlando, 2002

[CL01] L.L. Constantine and L.A.D. Lockwood, Structure and Style in Use Cases for User Interface Design, In M.van Harmelen, Hrsg., Object-oriented User Interface Design, 2001

[Co01] A. Cockburn: Writing Effective Use Cases, Addison Wesley, 2001

[De02] Ch. Denger: High Quality Requirements Specifications for Embedded Systems through Authoring Rules and Language Patterns, Diplomarbeit an der Universität Kaiserslautern, 2002

[De03] Ch. Denger: Inspection of High Level Statecharts, Technical Report at the Fraunhofer Institute for Experimental Software Engineering, to appear, 2003

[DOORS] http://www.telelogic.com

[Gl02] M. Glinz: Statecharts for Requirements Specification – As Simple As Possible, As Rich As Needed, ICSE-Workshop on Scenarios and State Machine Models, Algorithms and Tools, Orlando, 2002

[HG97] D. Harel, E. Gery: Executable Object Modeling with Statecharts, IEEE Computer, vol. 30, no.7, pg. 31-42, 1997

[HP98] D. Harel, M. Politit: Modeling Reactive Systems with StateCharts: The Statemate Approach, McGrawHill 1998

[HP02] F. Houdek, B. Paech: Das Türsteuergerät – eine Beispielspezifikation, IESE-Report Nr. 002.02/D, 2002

[HR02] P. Hruschka, Ch. Rupp: Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML, Hanser Verlag, 2002

[Ja95] I. Jacobson: The Use-Case Construct in Object-oriented Sofware Engineering. In J.M. Carroll, Hrsg., Scenario-based Design, Seiten 309-336. John Wiley & Sons, 1995.

[K+01] E.Kamsties, A. von Knethen, B. Paech: Structure of QUASAR Requirements Documents, IESE-Report No. 073.01/E, 2001

[Ki95] J.U. Kieß: Objektorientierte Modellierung von Automatisierungssystemen, Springer Verlag, 1995

[vK01] A. von Knethen: Change-oriented Requirements Traceability. Support for Evolution of Embedded Systems, PhD Theses in Experimental Software Engineering, vol. 9, Fraunhofer IRB Verlag, 2001

[vKP99] A. von Knethen, B. Paech: Überlegungen zum objektorientierten Entwurf eingebetteter Systeme, Workshop OMER, Tutzing, 1999

[vK+02] A. von Knethen, B. Paech, F. Kiediasch, F. Houdek: Systematic Requirements Recycling through Abstraction and Traceability, RE'02, pg. 273-281, 2002

[PM95] D. L. Parnas, J. Madey: Functional documents for computer systems, Science of Computer Programming, 25, pg. 41-61, May 1995

[PK00] B. Paech, E. Kamsties, Goal-Oriented Improvement of Requirements Processes, INSPIRE '00 (5th Int. Conference on SPI Research, Education &Training), London, pp. 169-182, September 2000

[RequisitePro] http://www.rational.com

[RG99] J. Ryser, M. Glinz: A Practical Approach to Validating and Testing Software Systems Using Scenarios, Quality Week Europe, 1999

[So02] S. Some: Beyond Scenarios: generating State Machine Models from Use Cases, ICSE-Workshop on Scenarios and State Machine Models, Algorithms and Tools, 2002

[UML] http://www.omg.org/uml/

[VV] VDI/VDE-Richtlinie 3694 Lasten/Pflichtenheft für den Einsatz von Automatisierungssystemen, 1991

[WS00] J. Whittle, J. Schumann: Generating Statechart Designs from Scenarios, ICSE 2000, Limerick, pg. 314-323

[WW02] M: Weber, J. Weisbrod: Requirements Engineering in Automotive Development – Experiences and Challenges, RE'02, pp.331-340, 2002