



Fraunhofer Institut
Experimentelles
Software Engineering

Richtlinien

Von Use Cases zu Statecharts in 7 Schritten



Autoren:

Christian Denger
Daniel Kerkow
Antje von Knethen
Maricel Medina Mora
Barbara Paech

Unterstützt durch das BMBF unter
dem Förderkennzeichen VFG0004A
(" QUASAR")

IESE-Report Nr. 086.02/D
Version 1.0
18. Dezember, 2002

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft. Das Institut überträgt innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen. Das Fraunhofer IESE steht unter der Leitung von Prof. Dr. Dieter Rombach Sauerwiesen 667661 Kaiserslautern

Abstract

Dieses Dokument beschreibt Richtlinien, um aus einem mit Use Cases beschriebenen Anforderungsdokument systematisch Statecharts zu erstellen. Die Richtlinien wurden anhand einer Fallstudie eines Türsteuerungsgerätes aus der Automobilindustrie evaluiert.

Schlagworte: Richtlinien, Use Cases, Statecharts, Rhapsody, Anforderungen, Modellierung

Inhaltsverzeichnis

1	Einleitung	1
1.1	Problem	1
1.2	Ziel	1
1.3	Unser Ansatz	2
1.4	Voraussetzungen/Vorbemerkung	2
2	Gestaltung der folgenden Richtlinien	4
2.1	Abkürzungen und Synonyme	4
2.2	Schreibkonventionen	4
2.3	Beschreibung eines Schrittes der Richtlinien	5
3	Anleitung zur Schrittweisen Ableitung der HLSC	7
3.1	Vorbereitung	7
3.1.1	Identifizieren von relevanten Use Cases	7
3.1.2	Identifizieren von speziellen UC Beziehungen (Unterbrechungen)	10
3.1.3	Auswahl monitorierter und kontrollierter Größen	11
3.1.4	Wertebereiche und Äquivalenzklassen identifizieren	12
3.1.5	Größen mit gleichem Wertebereich zusammenfassen	13
3.1.6	Modellieren von sowohl monitorierten als auch kontrollierten Größen	15
3.1.7	Berücksichtigen von Qualitätsaspekten	16
3.1.8	Diskussion: Komplexe Variablen	16
3.2	Schritt 1: Erstellung der Package-Struktur des Systems	17
3.3	Schritt 2: Erstellung des Klassendiagramm	18
3.3.1	Modellierung des Klassendiagramms	18
3.3.2	Modellierung von Assoziationen zwischen Klassen	21
3.3.3	Modellierung der Simulations- Umgebung	24
3.3.4	Diskussion: Bestimmung der Anzahl der modellierten Instanzen	25
3.4	Schritt 3: Modellierung interner Größen	26
3.5	Schritt 4: Modellierung der SC der monitorierten Größen	26
3.6	Schritt 5: Modellieren der SC der kontrollierten Größen	31
3.7	Schritt 6: Erstellen Sie die Use Case Statecharts (UCSCs)	32
3.7.1	Erstellung der HLSCs für die UC-Klassen	32
3.7.2	Modellierung der Ausnahmefälle der UC	35
3.8	Schritt 7: Verfeinern der Systemreaktion (insbesondere Umsetzung von Regeln)	38

4	Zusammenfassung und Ausblick	44
5	Anhang	45
6	Literatur	49

1 Einleitung

1.1 Problem

Moderne Kraftfahrzeuge werden durch eine zunehmend ansteigende Anzahl (20-80) von Steuergeräten (engl. electronic control units) kontrolliert. Das erfordert eine hohe Qualitätssicherung und somit höhere Standards für Anforderungsdokumente. Hinzu kommt, dass Fahrzeugserien häufig über mehrere Jahre hinweg in Kooperation zwischen Herstellern und Zulieferern entwickelt werden. Dies macht die Veränderbarkeit der Dokumente und die klare Definition von Schnittstellen erforderlich. Damit die Qualität erhöht werden kann, wird in der Regel werkzeuggestützt entwickelt, was einen problemlosen Übergang von frühen natürlichsprachlichen Anforderungsdokumenten zu grafischen Modellen, wie etwa Statecharts notwendig macht.

1.2 Ziel

Das Ziel unserer Forschungstätigkeit ist es, präzise und effiziente Richtlinien zur Erstellung und Verwendung von Anforderungsdokumenten für Steuergeräte zu definieren. Um eine effiziente Kommunikation und Qualitätssicherung zu ermöglichen, werden Anforderungen auf vier Abstraktionsebenen betrachtet:

- Systemanforderungen beschreiben die Perspektive des Herstellers und des Benutzers des Fahrzeugs.
- Die Systemspezifikation beschreibt die detaillierte Funktionalität des gesamten Systems (inklusive Hardware und Software), abstrahiert jedoch von Details der realen Sensoren, Aktuatoren und Benutzerschnittstellen.
- Softwareanforderungen beschreiben die Anforderungen an die Software als Teil des Gesamtsystems.
- Die Softwarespezifikation beschreibt die detaillierte Funktionalität der Software, die notwendig ist, um mit speziellen Sensoren und Aktuatoren zu interagieren.

Die Unterscheidung zwischen Anforderungen (auch Systemlastenheft) und Spezifikationen (auch Systempflichtenheft) ermöglichen dem Hersteller das Einbeziehen von Zulieferern auf allen Ebenen der System- und Softwareentwicklung ohne die Notwendigkeit internes Know-How zu veröffentlichen. Die Unterscheidung zwischen System- und Softwaredokumenten unterstützt klare Schnittstellen zwischen Hardware- und Softwareentwicklern. Darüber hinaus wird die Funktionalität unabhängig von deren Assoziation mit einzelnen Steuergeräten beschrieben.

1.3 Unser Ansatz

Um o.g. Ziele zu erreichen, werden die Dokumenten-Ebenen wie folgt realisiert:

- Funktionale Systemanforderungen werden mit *Use Cases* beschrieben. Diese sind bekannt für ihre gute Verständlichkeit. Deren Verwendung für eingebettete Systeme erfordert detaillierte Anleitung, wie Funktionalität mit Hilfe von *Use Cases* strukturiert werden kann.
- Funktionale Systemspezifikationen werden mit High level Statecharts beschrieben. Gemeint sind Statecharts, die das System auf einem hohen Abstraktionsgrad beschreiben. *High-level Statecharts* ermöglichen eine vollständige Beschreibung des Systems und können systematisch aus den *Use Cases* abgeleitet werden. Ihre Struktur spiegelt sowohl die Struktur der *Use Cases* als auch das Systemverhalten wieder, als eine Abbildung von monitorierten auf kontrollierten Größen. Abhängigkeiten zwischen Funktionsblöcken werden mittels Funktionsnetzen explizit gemacht.
- Funktionale Softwareanforderungen können aus den High level Statecharts durch Ersetzen der abstrakten monitorierten und kontrollierten Größen durch Sensoren und Aktuatoren abgeleitet werden.
- Funktionale Softwarespezifikationen können von den Softwareanforderungen abgeleitet werden, indem Details über die interne Softwareorganisation, wie z.B. Speicher hinzugefügt werden.

Die vorliegenden Richtlinien beziehen sich auf die Systemebene und geben Anleitung zum Ableiten der funktionalen Systemspezifikation aus funktionalen Systemanforderungen.

1.4 Voraussetzungen/Vorbemerkung

Die in diesem Bericht beschriebenen Richtlinien zur Ableitung von High Level Statecharts aus Use Case Diagrammen setzen voraus, dass die betrachteten *Use Cases* ebenfalls nach bestimmten Richtlinien erzeugt wurden. Die Richtlinien zur Definition von *Use Cases* stellen sicher, dass die Elemente der *Use Cases* (z.B. includes und extends-Beziehungen, Aktoren, monitorierte und kontrollierte Variablen) und deren Verwendung eindeutig definiert sind (siehe [DP02])

Diese Vorgabe schränkt die Anwendbarkeit der Richtlinien in einigen Punkten zwar ein, es ist aber von großer Bedeutung, dass die Ableitung der High Level Statecharts auf semantisch möglichst eindeutigen *Use Cases* aufsetzt. Die Richtlinien zur Erstellung der *Use Cases* werden benötigt, um die Semantik der Beziehungen zwischen *Use Case* zu definieren (includes und extends- Beziehungen). Weiter setzen die Richtlinien zur Ableitung der High Level Statecharts auf der Struktur zur textuellen Beschreibung von *Use Cases* auf, die in den Richtlinien zur Erstellung von *Use Cases* definiert ist. Ansonsten sind die Richtlinien unabhängig von den Richtlinien zur Erstellung von *Use Cases* einsetzbar.

Die Anwendung der Richtlinien wird anhand von Beispielen aus einer Fallstudie aus der Automobilindustrie erläutert. In dieser Fallstudie wird ein Türsteuergerät (TSG) beschrieben, das in einem Fahrzeug bestimmte Funktionen ausführt. Dazu gehören z.B. die Kontrolle des Türschlosses (Zentralverriegelung), die Ansteuerung der Außenspiegel und der Fensterscheiben, die elektronische Einstellung der Frontsitze sowie die Beleuchtung im Fahrzeuginnenraum. Die vollständige Spezifikation des TSG ist unter [HP01] nachzulesen.

Der weitere Bericht gliedert sich wie folgt: In Kapitel 2 werden zunächst einige Konventionen zur Beschreibung der Richtlinien festgelegt. In Kapitel 3 werden die einzelnen Schritte der Richtlinien im Detail beschrieben. Kapitel 4 fasst den Bericht zusammen und gibt einen Ausblick auf zukünftige Forschungsaufgaben.

2 Gestaltung der folgenden Richtlinien

2.1 Abkürzungen und Synonyme

State	Zustand
Event	Ereignis in der Umgebung des Systems oder im System selbst
SC	Statechart: Diagrammart, in der mehrere States und deren Übergänge dargestellt werden können.
UC	Use Case: Use Case Diagramme und textuelle Beschreibung.
UCKI	Use Case Klasse
UCSC	Use Case Statechart
Object Model Diagramm	Sowohl Klassen- als auch Objektdiagramm
HLSC	High Level Statecharts: Eine Technik zur modellbasierten Beschreibung von funktionalen Anforderungen auf der Ebene des Systempflichtenhefts.

2.2 Schreibkonventionen

Konventionen für die Verwendung von Präfixen von Bezeichnungen im vorliegenden Dokument:

Monitorierte und kontrollierte Größen beginnen mit Großbuchstaben:

- Monitorierte Größen: beginnen mit dem Präfix *M*, z.B.: *M_Door* (M wie monitored)
- Kontrollierte Größen: beginnen mit dem Präfix *C*, z.B.: *C_Part_Movement* (C wie controlled.)

Andere Statechart- Elemente beginnen mit Kleinbuchstaben:

- Events: beginnen mit dem Präfix *ev*, z.B.: *ev_Direct_Input*
- State: beginnen mit dem Präfix *s*, z.B.: *s_Waiting_Direct_Input*

Abkürzungen, die sich auf Code beziehen beginnen mit Kleinbuchstaben:

- Attribute beginnen mit dem Präfix *a*, z.B.: *a_Speed_Limit*
- Parameter-Variablen beginnen mit dem Präfix *p*, z.B.: *p_I*
- sonstige-Variablen beginnen mit dem Präfix *v*, z.B.: *v_I*
- Operationen beginnen mit dem Präfix *o*, z.B.: *o_Start_Movement*

2.3 Beschreibung eines Schrittes der Richtlinien

Die einzelnen Schritte der Richtlinien werden nach der folgenden Struktur dokumentiert:

Optionen

Möglichkeiten, wie ein bestimmter Schritt oder Unterschritt ausgeführt werden kann.

Option/Kriterien Matrix

Matrix die aufzeigt, welche der möglichen Optionen welche Auswahlkriterien erfüllt. Die verwendeten Auswahlkriterien sind:

Für den Benutzer der HLSCs:

- Verständlichkeit des Ergebnisses (**VerErg**), d.h. der HLSCs
- Verständlichkeit der Abbildung (**VerAbb**) UCs → HLSCs
- Wartbarkeit der HLSCs (**WarAuf**)

Für den Ersteller der HLSCs:

- Modellierungsaufwand (**ModAuf**) (Komplexität) der Option
- Verständlichkeit der Abbildung (**VerAbb**) UCs → HLSCs

Die folgende Matrix soll als Beispiel dienen:

	VerAbb	VerErg	ModAuf	WarAuf
Opt.1	+	+	+	-
Opt.2	+	-	0	-

Tabelle 1: Optionen/Kriterien Matrix

Ein „+“ bedeutet, dass die Option die Erfüllung des entsprechenden Kriteriums positiv beeinflusst. Zum Beispiel würde Opt. 1 zu einer guten Verständlichkeit der Abbildung zwischen UCs und HLSCs führen.

Ein „0“ bedeutet, dass diese Option die Erfüllung des entsprechenden Kriteriums weder positiv noch negativ beeinflusst oder dass bzgl. dieser Option nichts über die Erfüllung des Kriteriums ausgesagt werden kann.

Ein „-“ bedeutet, dass diese Option die Erfüllung des entsprechenden Kriteriums negativ beeinflusst. Opt.2 würde z.B. die Verständlichkeit der HLSC reduzieren.

Entscheidung

Option die gewählt wurde

Vorteile

Vorteile der gewählten Option

Nachteile

Nachteile der gewählten Option

Abgelehnte Optionen

Beschreibung der abgelehnten Optionen und Vor- und Nachteile dieser Optionen.

Anweisung

Hier werden konkrete Anweisungen gegeben, die in dem entsprechenden Schritt durchzuführen sind. Falls mehrere Optionen zur Auswahl stehen, werden hier lediglich die Anweisungen zur Ausführung der gewählten Option gegeben.

Rationale:

Gibt eine Begründung warum dieser Schritt nötig ist, bzw. warum eine bestimmte Option gewählt wurde.

Beispiel:

Ein Beispiel, das den Schritt erläutert. Diese Beispiele beschreiben, sofern dies möglich ist, stets die Situation vor und nach der Ausführung eines Guideline-Schrittes.

Die Paragraphen *Optionen*, *Option/Kriterien Matrix* und *Entscheidung* werden nur dann aufgeführt, wenn es in einem Schritt mehrere Optionen zur Ausführung des Schrittes gibt.

Die Beschreibung der verschiedenen Optionen und insbesondere die Dokumentation des Rationale, warum eine bestimmte Option gewählt wurde bzw. eine Aktivität durchgeführt werden muss, ermöglicht es, die Richtlinien leichter zu ändern. Dies ist dadurch begründet, dass durch das Rationale Entscheidungen nachvollziehbar werden. Weiter werden somit den Benutzern der Richtlinien Optionen zur Ausführung bestimmter Schritte bereitgestellt, die diese bei Bedarf einsetzen können. Dies macht die Richtlinien flexibler

3 Anleitung zur Schrittweisen Ableitung der HLSC

Die nachfolgenden Schritte leiten den Benutzer dieser Richtlinien systematisch dazu an, aus den UCs HLSCs abzuleiten. Mit Hilfe der Richtlinien wird das gesamte Verhalten eines Systems in den HLSCs abgebildet. Im Folgenden wird der Begriff „System“ verwendet, für eine Menge von Funktionsblöcken. Ein Funktionsblock beschreibt eine bestimmte Funktionalität des Systems, die Schnittstellen zu anderen Funktionsblöcken haben kann, aber bei Kenntnis dieser Schnittstellen unabhängig von anderen Funktionalitäten entwickelt werden kann. Die Modellierung des Gesamtverhaltens stellt sicher, dass alle möglichen Interferenzen zwischen den verschiedenen Systemteilen verstanden wurden. Auf diese Weise ist es auch möglich, parallel stattfindende Ereignisse (z.B. in gleichzeitig aktiven UCs) auch gleichzeitig zu visualisieren (simulieren).

Bei der Erstellung der HLSCs werden Ideen der Objekt-Orientierung, wie zum Beispiel Vererbungsprinzipien, verwendet. Diese erleichtern an vielen Stellen die Modellierung der HLSCs. Wir verwenden zur Modellierung der HLSCs das Tool Rhapsody in J, da dieses Tool objekt-orientierte Entwicklung unterstützt. Die Richtlinien können auch ohne die Verwendung von objekt-orientierten Modellierungstools verwendet werden, allerdings sind dann bestimmte Aspekte nicht zu modellieren. Möchte man nur die HLSCs aus den UCs ableiten, ohne die Herleitung eines Klassendiagramms, dann ist der erste und zweite Schritt der Richtlinien (siehe Kapitel 3.1 und 3.2) zu überspringen. Die Modellierung der HLSCs kann dann auch in anderen graphischen Modellierungstool durchgeführt werden, allerdings muss dann nach alternativen Modellierungen für die objekt-orientierten Prinzipien gesucht werden. Im Folgenden werden nun die einzelnen Schritte der Richtlinien beschrieben.

3.1 Vorbereitung

Bevor mit der HLSC- Modellierung begonnen wird, ist es empfehlenswert, die Elemente aus dem Systemlastenheft zu identifizieren, die für die Modellierung wichtig sind. Durch diese vorbereitenden Schritte werden dann die einzelnen Schritte der Richtlinien leichter anwendbar. Die UCs, die im Folgenden als Grundlage zur Beschreibung der einzelnen Schritte dienen können im Anhang nachgelesen werden.

3.1.1 Identifizieren von relevanten Use Cases

Betrachten Sie die UCs, welche zu den im Systemlastenheft beschriebenen Funktionsblöcke gehören. Wählen Sie, auf der Grundlage der folgenden Cha-

rakteristika, diejenigen UCs aus, die später als Klassen in einem Klassendiagramm modelliert werden:

- Beschreibt ein UC eigene Funktionalität, dann modellieren Sie den UC in den HLSCs. Ein UC beschreibt dann eigene Funktionalität, wenn in der textuellen UC-Beschreibung neben Include- Anweisungen noch zusätzlich Verhaltensbeschreibungen, Regeln oder Ausnahmen definiert werden.
- Inkludiert ein UC andere UCs, beschreibt jedoch keine eigene Funktionalität, dann wird der UC in den HLSCs nicht modelliert.

Fall 1: UC beschreibt eigene Funktionalität

Beschreibt ein UC eigene Funktionalität, so ist dieser für die Modellierung des Systemverhaltens wichtig (siehe Kapitel 3.3). Zur Modellierung eines solchen UC ergeben sich drei Optionen:

Optionen

- Option 1: Der UC wird als Klasse modelliert.
- Option 2: Es wird **eine** Klasse für alle UCs eines Funktionsblocks modelliert. Jeder UC wird als paralleler Zustand repräsentiert.
- Option 3: Es wird **eine** Klasse für alle UCs eines Funktionsblocks modelliert. Jeder UC wird als sequentieller Zustand repräsentiert.

	VerAbb	VerErg	ModAuf	WarAuf
Opt.1	+	+	-	+
Opt.2	+	0	0	-
Opt.3	0	0	0	-

Tabelle 2: Option/Kriterien Matrix Modellierung von UCS als Klasse

Entscheidung: Modellieren Sie die UCS als Klassen (Option 1)

Vorteile:

Durch diese Art der Abbildung ist eine sehr einfache Abbildung der UCs auf die SCs möglich, da sich jeder relevante UC als eine Komponente (Klasse) im Klassenmodell wiederfinden. Somit erreicht man eine sehr gute Verfolgbarkeit zwischen den UCs und den SCs (VerAbb+).

Weiter wird durch diese Art der Abbildung die Parallelität der UCs durch die Parallelität der Klassen modelliert. Es kann vorkommen, dass ein bestimmter UC mehrfach gleichzeitig ausgeführt werden muss. Durch die Modellierung der UCs als Klassen ist diese Anforderung sehr einfach durch Instanzenbildung der entsprechenden Klasse realisierbar (VerErg+).

Ein weiterer Vorteil ist die Möglichkeit, gemeinsame Funktionalität in einer Super-Klasse und somit in deren SC zu kapseln. Dies erleichtert dann Änderungen an dieser allgemeinen Funktionalität, da diese Änderungen zentral in der Super-Klasse vorgenommen werden können (WarAuf+).

Nachteile:

Der Nachteil dieser Option ist die explizite Modellierung der Kommunikation zwischen Instanzen von Klassen. Dies führt zu einem erhöhten Modellierungsaufwand in den HLSC-Modellen, ist aber nötig, wenn man das Systemverhalten simulieren möchte (ModAuf-). Weiterhin wird die Modellierung der Kommunikation dadurch erschwert, dass der Name der jeweiligen Instanz einer Klasse, die das Event empfangen soll, explizit angegeben werden muss.

Abgelehnte Option 2:

Auch diese Option erlaubt eine sehr einfache Abbildung zwischen den UCs und den HLSCs (VerAbb+). Die Modellierung der UCs als parallele States hätte den Vorteil, dass die Modellierung der Kommunikation zwischen den UCs sehr viel einfacher wäre. Bei parallelen Statecharts können beliebige Events verschickt und empfangen werden, ohne dass konkret das Ziel der Events angegeben werden muss (ModAuf+). Der Hauptnachteil dieser Option ist allerdings, dass die Sequentialität zwischen UCs explizit erzwungen werden muss, da die parallelen SC per Definition nicht sequentiell sind (ModAuf-). Weiter zeigt sich häufig die Notwendigkeit, Superklassen einzuführen, die eine vereinfachte Modellierung in den HLSC ermöglichen (siehe Kapitel 3.3). Superklassenbildung wird durch SC aber nicht unterstützt (WarAuf-).

Abgelehnte Option 3:

Auch bei dieser Option ist die Abbildung zwischen den UCs und den HLSCs sehr intuitiv (VerAbb+). Manche UCs stehen in sequentiellem Ablauf zueinander, was am einfachsten durch die Modellierung in sequentiellen SCs zu realisieren ist. Dadurch könnte man eine sehr gute Übersichtlichkeit der HLSCs erreichen, da der Leser schneller erkennt, welcher UC gerade aktiv ist (VerErg+). Dadurch wäre auch sehr einfach zu erkennen, welche UCs sich gegenseitig unterbrechen. Das Hauptproblem dieser Option ist darin zusehen, dass nicht alle UCs in sequentiellem Ablauf zueinander stehen, d.h. parallel laufende UCs könnten nicht modelliert werden (VerAbb-, VerErg-). Daher wäre eine Mischform aus Option 2 und Option 3 notwendig, um die Verhältnisse, die in den UCs beschrieben werden, korrekt nachzubilden. Dazu müssen sehr detaillierte Richtlinien definiert werden, die beschreiben, wann ein UC sequentiell und wann er parallel zu anderen UCs steht. Weiter ist eine Bildung von Superklassen auch bei dieser Option nicht möglich (WarAuf-).

Anweisung:

Notieren Sie sich alle UCs, die eigene Funktionalität besitzen, da diese später modelliert werden. Modellieren Sie diese UCs in den folgenden Schritten als Klassen und SCs dieser Klassen.

Rationale:

Option 1 wurde gewählt, da der Modellierung von parallel laufenden UCs eine sehr große Bedeutung beigemessen wird. Dies zeigt insbesondere die Beispielspezifikation, in der mehrere Fälle solcher paralleler UCs beschrieben werden.

Fall 2: UC beschreibt keine eigene Funktionalität

Beschreibt ein UC keine eigene Funktionalität, so ist er für die Modellierung des Systemverhaltens nur von untergeordneter Bedeutung. Es ergeben sich zwei Optionen zur Modellierung solcher UCs:

Optionen:

- Option 1: Der UC wird nicht modelliert.
- Option 2: Der UC wird ebenfalls als Klasse oder paralleler State modelliert.

	VerAbb	VerErg	ModAuf	WarAuf
Opt.1	-	+	+	+
Opt.2	+	-	-	-

Tabelle 3: Option/Kriterien Matrix: UC ohne eigene Funktionalität

Entscheidung: Der UC wird nicht modelliert (Option 1)

Vorteile:

Da diese UCs keinerlei Funktionalität beschreiben, sind sie auch für das HLSC-Modell irrelevant. Somit erspart man sich durch das Weglassen dieser UCs die Modellierung irrelevanter Aspekte (ModAuf+) und macht das Klassen- und das HLSC-Modell leichter verständlich und leichter änderbar (VerErg+; WarAuf+).

Nachteile:

Ein Nachteil dieser Option ist, dass keine 1:1-Abbildung zwischen den UCs und den Klassen bzw. deren HLSCs möglich ist (VerAbb-).

Abgelehnte Optionen:

Die Vorteile von Option 2 entsprechen den Nachteilen von Option 1 und umgekehrt.

Anweisung:

Modellieren Sie UC, die keine eigene Funktionalität besitzen, nicht in den HLSCs.

Rationale:

Option 1 wurde gewählt, da diese Alternative mehr Vorteile hat. Hat die Übereinstimmung der Struktur der UCs mit der des Klassendiagramms höchste Priorität, dann wäre Option 2 zu wählen.

3.1.2 Identifizieren von speziellen UC Beziehungen (Unterbrechungen)

Anweisung:

Erstellen Sie eine Tabelle mit allen Beziehungen, die zwischen den im vorhergehenden Schritt ausgewählten UCs bestehen, mit Ausnahme der includes- und

extends- Beziehungen. Hinweise darauf geben die Elemente „Beschreibung“ und „Ausnahmefälle“ der textuellen UC-Beschreibung.

Rationale:

Diese Tabelle wird Ihnen später helfen, Ausnahmefälle zu modellieren, die zu einem Übergang zwischen zwei UCs führen (Siehe 3.7.2). Durch die Tabelle wird weiter die Identifikation von Events (Transitionen) erleichtert, die diese Übergänge zwischen den UCs beschreiben.

Beispiel:

Tabelle 4 veranschaulicht solche UC Beziehungen. Diese Tabelle wurde basierend auf der Beispielspezifikation des Türsteuergerätes erzeugt. Die entsprechenden UCs sind im Anhang A beschrieben.

Use Case	Wechselt zu Use Case:	Wo Beschrieben
Steuere andere Fensterscheibe an	Steuere eigene Fensterscheibe an	Beschreibung, Ausnahmefälle
Steuere eigene Fensterscheibe an	Steuere andere Fensterscheibe an	Beschreibung, Ausnahmefälle
Steuere Fond Fensterscheibe an	Steuere andere Fensterscheibe an	Beschreibung, Ausnahmefälle
Steuere Fensterscheibe partiell an	Steuere Fensterscheibe total an	Beschreibung, Ausnahmefälle
Steuere Fensterscheibe total an	Steuere Fensterscheibe partiell an	Beschreibung, Ausnahmefälle

Tabelle 4: Beziehungen zwischen den Use Cases

3.1.3 Auswahl monitorierter und kontrollierter Größen

Anweisung:

Erstellen Sie eine Tabelle, die zum einen alle monitorierten Größen der Zeile „Eingänge“ und zum anderen alle kontrollierten Größen der Zeile „Ausgänge“ aus den in 3.1.1 ausgewählten UCs enthält. Ergänzen Sie die Tabelle mit dem Namen des UC, in dem die Größe auftritt. Ergänzen Sie im Falle einer monitorierten Größe den Namen der Aktoren, die diesen Input triggern können, falls diese zugeordnet werden können.

Rationale:

Die Tabelle wird Ihnen später helfen die Klassen und HLSCs der monitorierten und kontrollierten Größen zu modellieren.

Beispiel:

Die nachfolgend beschriebenen Tabellen zeigen auszugsweise die monitorierten und kontrollierten Größen des Funktionsblocks „Fenstersteuerung“ aus der Beispielspezifikation des Türsteuergerätes.

Monitorierte Größe	Use Case	Aktoren
Aktuelle Fensterscheibenbewegungsrichtung	Steuere Fensterscheibe total an, Steuere Fensterscheibe partiell an	Fahrer, Beifahrer, Fondinsasse

Tabelle 5: Beschreibung monitorierter Größen

Kontrollierte Größen	Use Case
Neue Fensterscheibenposition	Steuere Fensterscheibe total an, Steuere Fensterscheibe total an

Tabelle 6: Beschreibung monitorierter Größen

3.1.4 Wertebereiche und Äquivalenzklassen identifizieren

Anweisung:

Bestimmen Sie die Wertebereiche und die initialen Werte der monitorierten und kontrollierten Größen, die Sie in 3.1.3 bestimmt haben. Erweitern Sie dazu die Tabellen 3.1.3 um die Spalten „Wertebereich“ und „Initialwert“.

Fassen Sie ggf. Werte in den Wertebereichen zu Äquivalenzklassen zusammen. Äquivalenzklassen sind Wertebereiche, für die das System das gleiche Verhalten aufweist. Verändern Sie hierzu den Eintrag in der Spalte Wertebereich der betroffenen monitorierten und kontrollierten Variablen. Die Initialwerte bestimmen den Wert der Variablen bei der Initialisierung des Systems.

Kennzeichnen Sie alle die Größen, deren Wertebereich nicht in sinnvolle Äquivalenzklassen aufgeteilt werden kann, als interne Größen. Notieren Sie dazu in der Spalte „Wertebereich“ den tatsächlichen Wertebereich der Größe eingeschlossen in „[.]“, zum Beispiel könnte die Außentemperatur eine monitorierte Größe sein deren Wert zwischen -40 und +50 Grad Celsius liegen kann. Lässt sich der Wertebereich nicht aus Äquivalenzklassen abbilden so muss der Wert der Größe als Attribut gespeichert werden. In der Tabelle muss der Wertebereich in „[.]“ eingeschlossen werden, um diesen als internen Wertebereich zu kennzeichnen, d.h. im obigen Beispiel: [-40, +50].

Rationale:

Dieser Schritt ist wichtig, da sich der Modellierer bereits jetzt Gedanken zu den Wertebereichen der Variablen machen muss und somit die Abbildung der Wertebereiche auf Zustände in den HLSCs erleichtert wird. Weiter ist die Bildung von Äquivalenzklassen nötig, um kontinuierliche Wertebereiche auf diskrete Wertebereiche abzubilden. Somit modelliert man später nur die für das Systemverhalten relevanten Wertebereiche der Variablen.

Beispiel:

Das folgende Beispiel zeigt die erweiterte Tabelle aus 3.1.3.

Monitorierte Größe	Use Case	Aktoren	Wertebereich	Initialwert
Aktuelle Fenster-scheibenbewegung	Steuere Fenster-scheibe total an, Steuere Fenster-scheibe partiell an	Fahrer, Bei-fahrer, Fond-insasse	keine Bewe-gung; hoch; runter	keine Bewe-gung

Tabelle 7: Erweiterte Tabelle zur Beschreibung monitorierte Variablen

Als Beispiel für die Bildung von Äquivalenzklassen dient die Variable Geschwindigkeit. Diese kann z.B. einen beliebigen Wert zw. 0 und 250km/h annehmen. Das System verhält sich bei Geschwindigkeiten < 5 km/h anders als bei Geschwindigkeiten ≥ 5 km/h. Somit sind für das System relevanten Äquivalenzklassen z.B. „Unter Limit“ (< 5 km/h) und „Über Limit“ (≥ 5 km/h), da nur diese beiden Wertebereiche eine unterschiedliche Reaktion des Systems hervorrufen.

3.1.5 Größen mit gleichem Wertebereich zusammenfassen**Anweisung:**

Fassen Sie Größen mit dem gleichen Wertebereich zu einem Typ zusammen. Ergänzen Sie hierzu die Tabelle aus 3.1.3 um die Spalte „Zusammengefasste Größen“. Ermitteln Sie die Variablen, die nach 3.1.4 den gleichen Wertebereich besitzen. Fügen Sie diese Variablen zu einer neuen monitorierten oder kontrollierten Variable zusammen und ergänzen Sie die Tabelle um diese neue Variable. Beschreiben Sie in der Spalte „Zusammengefasste Größen“ die Variablen die von der „neuen“ Variable zusammengefasst werden.

Beachten Sie, dass sich Schritt 3.1.4 und Schritt 3.1.5 gegenseitig beeinflussen können. Es ist möglich, dass sich durch das Zusammenfassen von Größen mit gleichen Wertebereichen neue Wertebereiche und somit neue Äquivalenzklassen ergeben. Daher sollten die Schritte 3.1.4 und 3.1.5 iterativ durchgeführt werden.

Rationale:

Diese Art der Vereinfachung ist eine Frage der Ökonomie, d.h. vor allem wesentliche Unterschiede der Variablen sollen in den HLSC modelliert werden.

Durch die Zusammenfassung von zusammengehörenden Größen wird die Anzahl der später zu bildenden Klassen und HLSCs verringert. Dadurch wird wiederum die Änderbarkeit des Systems verbessert.

Beispiel:

Die folgende Tabelle zeigt beispielhaft mehrere monitorierte Größen nach Schritt 3.1.4

Monitorierte Größe	Use Case	Wertebereich	Initialwert
Abstand_Sitz_Lenkrad	Stelle Sitz gemäß direkter Positionseingabe ein	stop, vor, zurück	stop
Sitzfläche_vorne	Stelle Sitz gemäß direkter Positionseingabe ein	stop, heben, senken	stop
Sitzfläche_hinten	Stelle Sitz gemäß direkter Positionseingabe ein	stop, heben, senken	stop
Schalung	Stelle Sitz gemäß direkter Positionseingabe ein	stop, enger, weiter	stop
Lehnenwinkel	Stelle Sitz gemäß direkter Positionseingabe ein	stop, steiler, flacher	stop

Tabelle 8: Beispiel für monitorierte Größen mit gleichem Wertebereich

Durch Schritt 3.1.5 wird deutlich, dass man diese Variablen in einer abstrakteren Größe zusammenfassen kann. Die nachfolgende Tabelle zeigt das Ergebnis dieser Zusammenfassung, d.h. die Tabelle nach der Durchführung von Schritt 3.1.5

Monitorierte Größe	Use Case	Wertebereich	Initialwert	Zusammengefasste Größen
Direkte Positionseingabe	Stelle Sitz gemäß direkter Positionseingabe ein	stop, einschränken, entspannen	stop	Abstand_Sitz_Lenkrad, Sitzfläche_vorne, Sitzfläche_hinten, Schalung, Lehnenwinkel

Tabelle 9: Zusammengefasste Größen

Es ist zu beachten, dass Tabelle 9 sowohl die Zusammenfassung von monitorierten Größen zeigt, als auch die Bildung von „Äquivalenzklassen“. Der Wertebereich der Größe „Direkte Positionseingabe“ ergibt sich aus der Tatsache, dass das System bei jeder Eingabe der Einzelgrößen entweder eine entspannende Bewegung (entspricht zurück, senken, weiter, flacher), eine einschränkende Bewegung (vor, heben, enger, steiler) oder keine Bewegung (stop) durchführt.

3.1.6 Modellieren von sowohl monitorierten als auch kontrollierten Größen

In eingebetteten Systemen kommt es häufig vor, dass Größen sowohl monitoriert als auch kontrolliert werden. Es ergeben sich zwei Möglichkeiten, solche Größen zu modellieren:

Optionen:

- Option 1: Solche Größen werden als eine Größe modelliert (gleichzeitig monitoriert und kontrolliert).
- Option 2: Solche Größen werden zweimal modelliert (sowohl monitoriert als auch kontrolliert).

	VerAbb	VerErg	ModAuf	WarAuf
Opt.1	0	-	+	0
Opt.2	0	+	-	-

Tabelle 10: Option/Kriterien Matrix monitorierte/kontrollierte Größen

Entscheidung: Solche Größen werden zweimal modelliert (Option 2).

Vorteile:

Nur durch die doppelte Modellierung ist eine Trennung zwischen Umgebungsgrößen, die vom System monitoriert werden und solchen die kontrolliert werden möglich, was dem Variablenmodell von Parnas entspricht, das diesen Richtlinien zugrunde liegt (VerErg+)

Nachteile:

Da die Variablen doppelt modelliert werden, führt dies zu mehr Klassen und HLSC und somit zu höherem Modellierungsaufwand und Wartungsaufwand (siehe 3.3.1) (ModAuf-, WarAuf-).

Abgelehnte Option:

Die Modellierung von sowohl monitorierten als auch kontrollierten Größen durch eine Größe, die gleichzeitig beide Eigenschaften hat würde zwar zu einer geringeren Anzahl von Klassen und HLSCs führen (ModAuf+), allerdings müsste dann z.B. eine kontrollierte Variablen Input in das System liefern, was nicht dem intuitiven Verständnis dieser Größen entspricht (VerErg-).

Anweisung:

Modellieren Sie die Größen, die sowohl monitoriert als auch kontrolliert werden, doppelt. Einmal als monitorierte Größen und einmal als kontrollierte Größen.

Rationale:

Trotz eines höheren Modellierungsaufwands ist Option 2 zu wählen, da nur so eine korrekte Umsetzung des 4 Variablenmodells ermöglicht wird.

Beispiel:

Vergleichen Sie als Beispiel Kapitel 3.3.3, in dem die Klassen für das Simulations- Environment beschrieben werden.

3.1.7 Berücksichtigen von Qualitätsaspekten**Anweisung:**

Beschreiben Sie in den UCs spezifizierte Qualitätsaspekte in den HLSCs mit Hilfe von Kommentaren. Beachten Sie, dass die Qualitätskriterien jedoch nicht durch die Modellierung erfüllt werden, d.h. es ist nicht möglich die Art und Weise der Realisierung der Qualitätskriterien in den HLSCs zu beschreiben.

Rationale:

Auf dieser Abstraktionsebene ist es noch nicht sinnvoll die konkrete Realisierung solcher nichtfunktionaler Anforderungen zu beschreiben.

Beispiel:

Ein Qualitätskriterium der Beispielspezifikation ist, dass ein Fenster innerhalb von 3 sec in der unteren Position sein muss. Die konkrete Realisierung dieser Anforderung kann nicht in den HLSCs überprüft werden, aber es ist möglich die Anforderung auch in den HLSCs z.B. mit Hilfe von Kommentaren zu beschreiben.

3.1.8 Diskussion: Komplexe Variablen

Neben den vorgestellten Überlegungen wurde weiter angedacht, ob über die normalen Variablen hinaus die Modellierung komplexer Variablen sinnvoll sein könnte. Diese komplexen Variablen könnten dann z.B. Operationen beinhalten, die den zusammengefassten Variablen gemein sind.

Diese Überlegungen basieren auf der Tatsache, dass es nicht immer sinnvoll ist z.B. eine monitorierte Größe als Klasse zu modellieren. In manchen Fällen ist nur der Wert einer Größe von Bedeutung, nicht aber ein Wechsel des Werts, der dem System aktiv mitgeteilt werden muss. In einem solchen Fall würde die Modellierung eines Attributs ausreichen, das innerhalb einer komplexen Variable definiert werden könnte.

Die Definition einer abstrakteren Größe, die den direkt monitorierten Größen übergeordnet wäre (z. B. Tür, die einen Verriegelungszustand, einen Öffnungszustand und den Status der Kindersicherung enthält), unterstützt die Verständlichkeit des Systems bei einer sehr großen Menge von direkt monitorierten Größen.

Natürlich ist eine solche Überlegung nur sinnvoll für die Zusammenfassung atomarer Größen, d.h. Größen die nicht aus anderen Größen zusammengesetzt sind. Hier liegt aber eines der Probleme. Es muss eindeutige Richtlinien geben, wann eine Größe als atomar und wann sie als komplex gilt. Ein weiterer offener Diskussionspunkt ist die Problematik, dass es durch die Bildung solcher komplexen Größen dazu kommt, dass monitorierte und kontrollierte Systemelemente in einer solchen Größe zusammengefasst werden. Die Frage, die sich dann stellt, ist: handelt es sich bei der komplexen Größe um eine monitorierte oder kontrollierte Größe?

Um diese Fragen beantworten zu können, sind noch weitere Untersuchungen notwendig. Daher werden wir auf diesen Punkt nicht weiter eingehen. Der Leser muss selbst entscheiden, ob in seinem konkreten Fall die Bildung solcher Variablen sinnvoll ist oder nicht.

3.2 Schritt 1: Erstellung der Package-Struktur des Systems

Anweisung:

Erstellen Sie für jeden Funktionsblock des Systemlastenhefts ein Package.

Rationale:

Die Package-Struktur des Systems verdeutlicht sehr schnell und einfach die vom System realisierten Funktionsblöcke.

Beispiel:

In der Beispieldokumentation konnten die folgende Funktionsblöcke identifiziert werden:

- Sitzsteuerung
- Fenstersteuerung
- Außenspiegeleinstellung
- Türschloss
- Innenraumbelichtung

Daraus ergibt sich folgende Package-Struktur für das System:

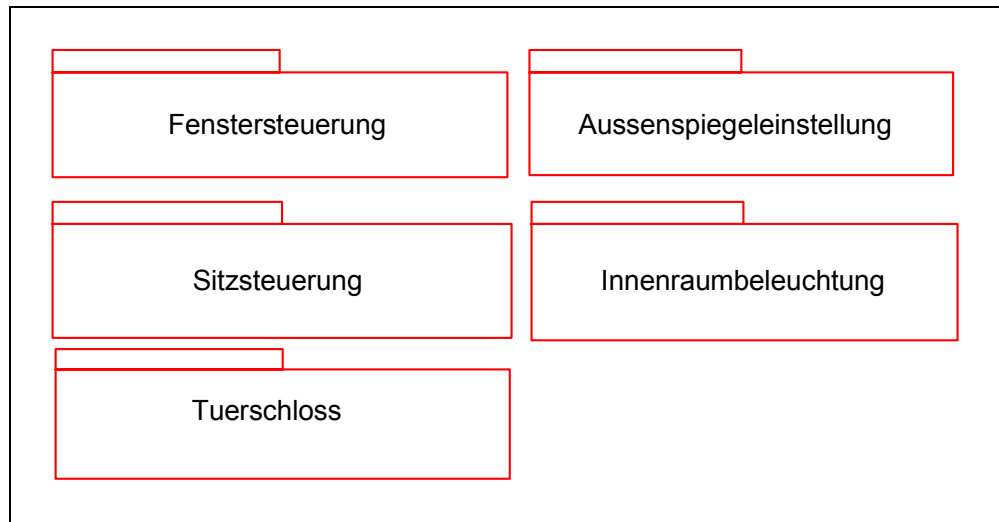


Abbildung 1: Package- Struktur des Beispielsystems

3.3 Schritt 2: Erstellung des Klassendiagramm

In diesem Schritt und den zugehörigen Unterschritten wird zunächst ein Klassendiagramm erstellt. Die Erstellung dieses Klassendiagramms ist aus objekt-orientierten Gesichtspunkten nötig, da SCs das dynamische Verhalten von Klassen repräsentieren. Daher müssen zunächst Klassen definiert werden, für die dann in den weiteren Schritten HLSCs modelliert werden können. Die folgenden Unterschritte leiten Sie systematisch dazu an, Klassen und Assoziationen zwischen diesen Klassen zu erstellen.

Die Richtlinien sind so aufgebaut, dass für jedes Package des Systems nun das Klassendiagramm und im weiteren die zugehörigen HLSCs modelliert werden. Das heißt die im Folgenden beschriebenen Schritte beziehen sich alle auf ein Package und sind dann für die restlichen Packages zu wiederholen, um eine Modellierung des gesamten Systems zu erreichen. Alle folgenden Ausführungen beziehen sich auf das Package „Fenstersteuerung“, d.h. die Richtlinien sind am Beispiel dieses Package beschrieben.

3.3.1 Modellierung des Klassendiagramms

Anweisung:

Erstellen Sie ein Klassendiagramm pro Funktionsblock und fügen Sie die Klassen des zu modellierenden Systems ein. Folgende Elemente der UCs sind als Klassen hinzuzufügen:

- 1.) Für jeden Aktor eine Inputklasse vom Typ „Aktor_Name_Input“
- 2.) Alle in 3.1.3 identifizierten monitorierten Größen der UCs eines Funktionsblocks, die nicht einem Aktor der Umgebung zugeordnet werden konnten. Es ist möglich, dass Sie während der Durchführung von Schritt 1 und 2 feststellen, dass die gleiche monitorierte Variable in mehreren Packages verwendet wird. In einem solchen Fall sollten Sie, um Redundanzen zu vermeiden, diese Variable in nur einem Package modellieren und von allen weiteren Packages auf die Klasse dieser Variable zugreifen.
- 3.) Alle in 3.1.3 identifizierten kontrollierten Größen, der UCs eines Funktionsblocks.
- 4.) Eine Klasse für jeden in 3.1.1 identifizierten UCs des Funktionsblocks.
- 5.) Eine Timer-Klasse.

Rationale:

Aus objekt-orientierter Sicht müssen Klassen modelliert werden bevor SCs modelliert werden können. Dieses Klassendiagramm vermittelt auch einen sehr guten Überblick über das Gesamtsystem und spiegelt die Struktur der UCs wieder.

Die Timer-Klasse wird benötigt, um zeitabhängiges dynamisches Verhalten zu modellieren und um kontinuierlich zuüberprüfende Bedingungen in den HLSCs zu realisieren.

Die Zuordnung der möglichen Eingaben in das System zu den entsprechenden Aktoren, die diese Eingaben tätigen können, fördert die Verständlichkeit der HLSCs. Der Leser kann sofort erkennen, welche Aktoren mit welchen UCs interagieren können.

Beispiel:

Abbildung 2 zeigt das UC-Diagramm des Funktionsblocks „Fenstersteuerung“ der Beispieldokumentation.

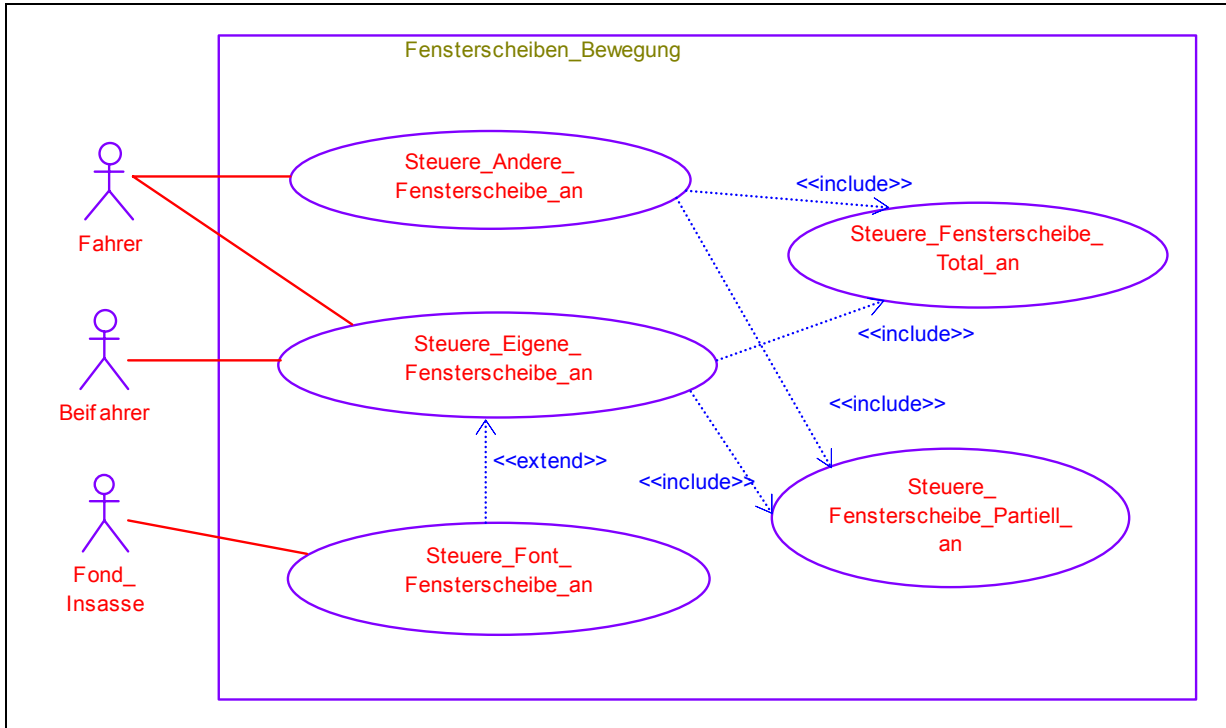


Abbildung 2: UC-Diagramm des Funktionsblocks „Fenstersteuerung“

Aus dem in Abbildung 2 dargestellten UC-Diagramm und nach der Durchführung der vorbereitenden Schritte 3.1.1, 3.1.3, 3.1.4, 3.1.5 und 3.1.6, in denen die textuelle Beschreibung der UCs analysiert wird, konnten die folgenden monitorierten und kontrollierten Größen ermittelt werden:

Monitorierte Größen:

- Fenster_Position_Ist
- Kindersicherung
- Aktuelle_Fensterscheibenansteuerung

Kontrollierte Größen:

- Fenster_Position_Soll

Diese Informationen führen dann zu dem in Abbildung 3 dargestellten Klassendiagramm. Im oberen Bereich sind vier Klassen für monitorierte Variablen (Fahrer_Input, Beifahrer_Input, Fond_Insassen_Input und Kindersicherung) modelliert. Durch die Klassen Fahrer_Input, Beifahrer_Input, Fond_Insassen_Input werden alle Eingaben in das System modelliert, die durch diese Aktoren getä-

tigt werden können, in diesem Fall die monitorierte Größe Aktuelle_Fensterscheibenansteuerung.

Im unteren Bereich ist die kontrollierte Variable (Fenster_Position_Soll) modelliert. Links davon wird die Simulationsumgebung durch die Klasse „Prozess-Kette“ modelliert (siehe dazu 3.3.3) sowie die monitorierte Variable „Fenster_Position_Ist“. Die fünf Klassen in der Mitte des Diagramms repräsentieren die UCs, die eigene Funktionalität beschreiben. Zur Bestimmung der Anzahl der Instanzen der einzelnen Klassen beachten Sie bitte die Diskussion in Abschnitt 3.3.4.

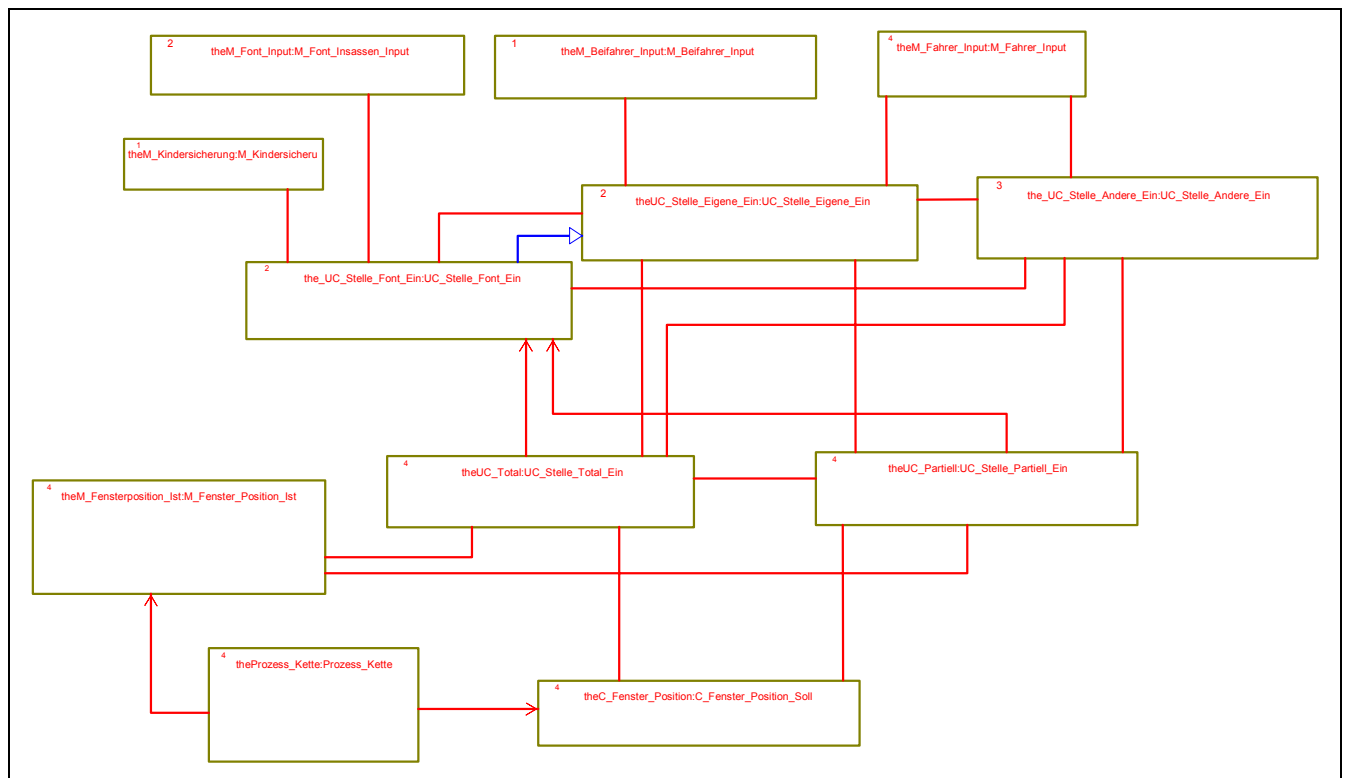


Abbildung 3: Klassendiagramm des Funktionsblocks "Fenstersteuerung"

3.3.2 Modellierung von Assoziationen zwischen Klassen

In diesem Schritt werden die Zusammenhänge zwischen den verschiedenen UC Klassen sowie die Zusammenhänge zwischen den UC Klassen und den Klassen der monitorierten und kontrollierten Größen modelliert. Solche Relationen werden als Assoziationen zwischen den verschiedenen Klassen, bzw. zwischen den Instanzen der Klassen repräsentiert.

Anweisung:

Erstellen Sie Assoziationen zwischen den verschiedenen Klassen nach folgendem Muster:

- Eine Klasse einer monitorierten Größe hat immer eine bidirektionale Assoziation zu den UC Klassen, die diese Größe als Eingang benötigen.
- Eine UC Klasse hat immer eine gerichtete Assoziation zu den Klassen der von diesem UC kontrollierten Größen .
- Eine UC Klasse hat eine bidirektionale Assoziation zu einer anderen UC Klasse, wenn sich die beiden UCs, die durch die Klassen repräsentiert werden, gegenseitig unterbrechen. Betrachten Sie hierzu die in dem vorbereitenden Schritt 3.1.2 erstellte Tabelle.
- Eine UC Klasse hat eine gerichtete Assoziation zu den Klassen der von diesem UC unterbrochenen UCs.
- Eine UC Klasse hat eine bidirektionale Assoziation zu einer anderen UC Klasse, wenn zwischen den entsprechenden UCs eine „Includes“ -Beziehung besteht.
- Eine UC Klasse hat eine Vererbungsbeziehung zu einer anderen UC Klasse, wenn zwischen den entsprechenden UCs eine „Extends“ - Beziehung besteht. Modellieren Sie in einem solchen Fall den UC, der erweitert wird, als Super- Klasse und den UC, der die Erweiterung beschreibt, als Sub- Klasse.

Rationale:

Die erste Assoziation ist notwendig, damit die monitorierte Größe eine Änderung ihres Werts an die UC Klasse weitergeben kann, bzw. deren Werte abgefragt werden können. Somit wird bei der Modellierung der Assoziationen noch keine Unterscheidung in aktive und passive Sensoren getroffen. In der weiteren Modellierung muss dann festgelegt werden, wie die Kommunikation zwischen monitorierten Größen und den Klassen der UCs realisiert wird (vgl. dazu Kapitel 3.5).

Die zweite Assoziation ist notwendig, damit die UC Klasse die kontrollierten Größen beeinflussen kann. Da die kontrollierten Größen keine Werte an das System liefern können, ist diese Assoziation unidirektional.

Die dritte und vierte Assoziation ist nötig, damit sich UC Klassen gegenseitig Events schicken können, um etwa eine Unterbrechung zu initiieren oder eine Rückmeldung zu geben, dass eine gewisse Funktionalität erbracht wurde.

Durch die fünfte Assoziation ist es möglich, den UCs gemeinsame Funktionalität in einer Superklasse zu modellieren. Diese Funktionalität wird dann an die Unterklassen vererbt. Diese Art der Modellierung kapselt die Funktionalität von mehreren Klassen an einer zentralen Stelle und führt somit zu einer leichteren Änderbarkeit des Systems (Kapselung von Gemeinsamkeiten in einer Klasse) und zu einer verbesserten Modellierungsökonomie. Man könnte auch UCs, die

Gemeinsamkeiten kapseln, als gleichberechtigte Klassen modellieren, dann verliert man aber die oben genannten Vorteile.

Beispiel:

Das folgende Beispiel illustriert die beschriebenen Beziehungen zwischen UCs. Betrachten Sie noch einmal Abbildung 2. Dort inkludiert der UC „Steuere Eigene Fensterscheibe an“ die UCs „Steuere Fensterscheibe Partiiell an“ und „Steuere Fensterscheibe Total an“. Weiter wird der UC „Steuere Eigene Fensterscheibe an“ erweitert durch den UC „Steuere Fond Fensterscheibe an“, d.h. die den beiden UCs gemeinsame Funktionalität wird im UC „Steuere Eigene Fensterscheibe an“ gekapselt.

Die Abbildung dieser Beziehungen zwischen den UCs werden in der folgenden Abbildung auf die entsprechenden Beziehungen zwischen den UC Klassen abgebildet:

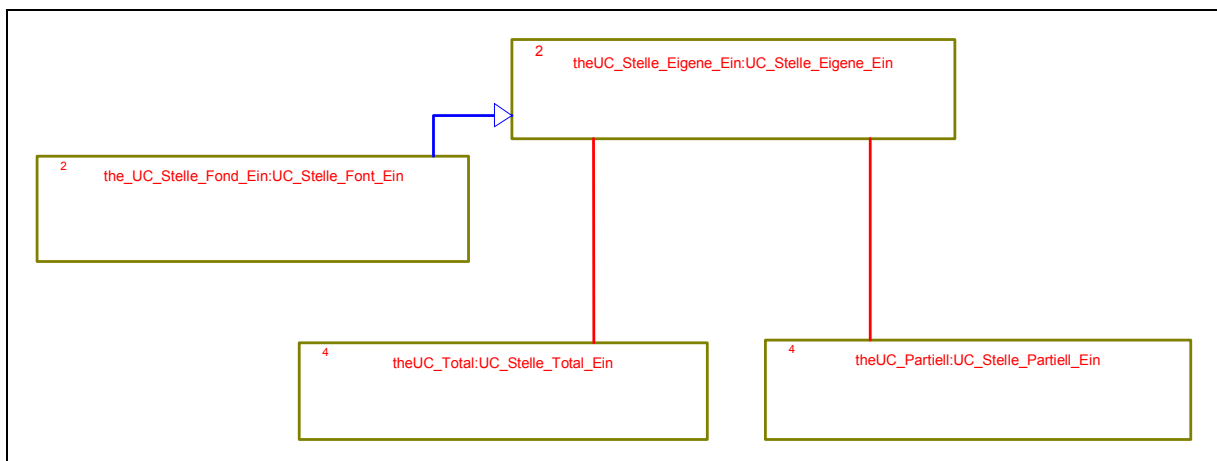


Abbildung 4: Assoziationen zwischen Klassen

Die Klasse „UC_Stelle_Eigene_Ein“ entspricht dem UC „Steuere Eigene Fensterscheibe an“ aus Abbildung 2. Die beiden Assoziationen zu den Klassen „UC_Stelle_Total_Ein“ und „UC_Stelle_Partiiell_Ein“, entsprechen den „Includes“-Beziehungen des UC „Steuere Eigene Fensterscheibe an“ zu den UCs „Steuere Fensterscheibe Total an“ und „Steuere Fensterscheibe Partiiell an“.

Weiter ist aus Abbildung 2 ersichtlich, dass der UC „Steuere Eigene Fensterscheibe an“ vom UC „Steuere Fond Fensterscheibe an“ erweitert wird („Extends-Beziehung“). Daher wird die Klasse „UC_Stelle_Eigene_Ein“ als Generalisierung der Klasse „UC_Stelle_Fond_Ein“ modelliert.

3.3.3 Modellierung der Simulations- Umgebung

Anweisung:

Erstellen Sie eine Klasse für die Simulations- Umgebung. Diese Klasse repräsentiert die reale Welt und in ihr ablaufende Prozessketten.

Rationale:

Die Modellierung der Prozesskette ist notwendig, um die Ablauffähigkeit des HLSC Modells zu garantieren, d.h., um dessen Simulierbarkeit sicher zu stellen. Für die Simulation können in der Klasse der Prozesskette Ereignisse simuliert werden, die in der realen Welt geschehen.

Es ist klar, dass die Beschreibung der Umgebung zusätzlichen Modellierungsaufwand bedeutet, aber es ist notwendig, um die Umgebung des Systems korrekt zu simulieren. Achten Sie darauf, dass die Klasse der Prozesskette bei einer späteren Implementierung wieder gelöscht werden muss.

Beispiel:

Abbildung 5 zeigt die Prozesskette im HLSC-Modell der Fenstersteuerung. Die Klasse „C_Fenster_Position_Soll“ repräsentiert die kontrollierte Variable „aktuelle Fensterposition“ des UC. Da diese kontrollierte Variable gleichzeitig auch monitoriert werden muss, d.h. ebenfalls eine monitorierte Variable darstellt benötigt man eine zweite Klasse im Klassendiagramm, welche die aktuelle Fensterposition als Eingang repräsentiert. In der Abbildung ist dies die Klasse „M_Fenster_Position_Ist“.

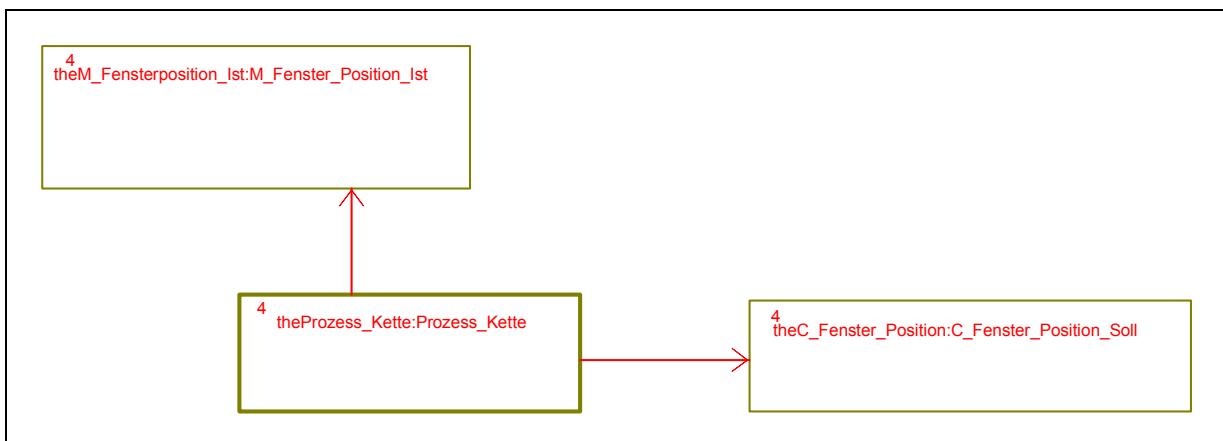


Abbildung 5: Simulation der Prozesskette

Weiter wird eine Klasse benötigt, die die Prozesskette repräsentiert. Diese sorgt dafür, dass Veränderungen in der kontrollierten Variablen der monitorierten Variable bekannt gemacht werden. Weiter kann es passieren, dass ein Gegenstand im Fenster eingeklemmt wird. Ein entsprechendes Event kann dann in der Klasse „Prozess_Kette“ erzeugt werden. Diese führt in einem solchen Fall kein

Update der Werte der Ist-Position mehr durch. Das System stellt dann automatisch fest, dass sich die Fensterposition nicht verändert obwohl der Motor angesteuert wird und aktiviert dann den Einklemmschutz.

3.3.4 Diskussion: Bestimmung der Anzahl der modellierten Instanzen

Eine offene Frage, die in weiteren Untersuchungen geklärt werden muss ist, wie man die Anzahl der erforderlichen Instanzen der Klassen für die UCs, und die Instanzen der Klassen der monitorierten und die kontrollierten Größen bestimmen kann.

Für die UC-Klassen konnten bisher zwei Regeln ermittelt werden:

- Bei UC-Klassen, die direkt mit den monitorierten Variablen, also deren Klassen assoziiert sind, ist die Anzahl der Instanzen der UC-Klassen gleich der Anzahl der Instanzen der monitorierten Größe, die mit diesem UC kommunizieren.
- Bei UC-Klassen, die direkt mit den kontrollierten Größen assoziiert sind, ist die Anzahl der Instanzen der UC Klasse gleich der Anzahl der Instanzen der kontrollierten Größe.

Diese Regeln repräsentieren erste Vermutungen über die systematische Ermittlung der nötigen Instanzen, sie konnten aber noch nicht validiert werden.

Da die beiden oben genannten Regeln voraussetzen, dass die Anzahl der Instanzen der Klassen für die monitorierten und kontrollierten Größen bekannt ist, muss in einem weiteren Schritt analysiert werden, wie man systematisch zur Anzahl dieser Instanzen gelangen kann.

Bei der Modellierung des Klassendiagramms der Fenstersteuerung aus der Beispielspezifikation (siehe Anhang und [HP01]) ergeben sich zwei Modellierungsalternativen, die die Gesamtanzahl der Instanzen beeinflussen. Es ist zum einen möglich ein HLSC Modell für je ein Fenster zu erzeugen, welches die gesamte Funktionalität beschreibt und das dann später in allen vier Fenstern eines Fahrzeuges eingebaut werden muss. D.h. das HLSC Modell muss vier mal instantiiert, um die gesamte Funktionalität des Funktionsblocks zu beschreiben. Alternativ dazu ist es möglich, ein Modell zu bilden, das mit Hilfe von Instanzen vier Fenster gleichzeitig modelliert. Je nach Eingabe der Aktoren, werden dann die entsprechenden Instanzen der Fenster angesteuert. Die bisherigen Erfahrungen haben gezeigt, dass bei der zweiten Alternative insgesamt eine Instanz weniger gebildet werden muss, da der Fall, dass beim Fahrerfenster nur ein Eingang (in allen anderen Fenstern sind es zwei Eingänge) vorhanden ist, von vorne herein berücksichtigt werden kann. Bei der ersten Alternative muss man stets von der maximalen Funktionalität ausgehen, da das Modell für jedes Fenster gültig sein

muss, d.h. bei der Modellierung des Gesamtverhaltens (also alle 4 Fenster) würden auch beim Fahrerfenster zwei Eingänge instantiiert werden.

3.4 Schritt 3: Modellierung interner Größen

In diesem Schritt werden alle Größen bearbeitet, die in dem vorbereitenden Schritt 3.1.4 als „interne“ Größe markiert wurden.

Anweisung:

Modellieren Sie die möglichen Werte der internen Größen als Attribute der Klassen, die diese Größen repräsentieren. Die entsprechenden Klassen haben Sie in Schritt 2 modelliert.

Rationale:

Für interne Größen können keine sinnvollen States festgelegt werden, da der Wertebereich dieser Größen nicht auf entsprechende Zustände abgebildet werden kann.

Beispiel:

Die kontrollierte Größe „Gespeicherte Sitzposition“ aus dem Funktionsblock „Sitzsteuerung“ der Beispielspezifikation dient dazu die Koordinaten der Sitzeinstellung eines Benutzers zu speichern. Der Wertebereich dieser Variable ist kontinuierlich und kann nicht in sinnvolle Äquivalenzklassen unterteilt werden. Daher ist es nicht möglich ein SC für die Klasse „C_Memory“ (repräsentiert die Größe „Gespeicherte Sitzposition“) zu erzeugen, welches diesen Wertebereich auf States abbildet. In diesem Fall muss ein Attribut, z.B. „a_Speicher_Position“, angelegt werden, das die gespeicherten Koordinaten repräsentiert.

3.5 Schritt 4: Modellierung der SC der monitorierten Größen

Die Schritt 4 und Schritt 5 (siehe dazu 3.5 und 3.6) betrachten die Größen, die in dem vorbereitenden Schritt 3.1.4 **nicht** als interne Größen klassifiziert wurden.

Zur Modellierung der monitorierten und kontrollierten Größen (siehe dazu 3.6) im HLSC-Modell ist weiter folgendes zu beachten. Die Größen (Variablen) werden als Teil des Systems betrachtet. Somit fungiert die Modellierung der jeweiligen Größe im HLSC als Puffer der tatsächlichen externen Größe. Über die Modellierung der Zustände der monitorierten und kontrollierten Größen können Ereignisse aus der Umgebung gepuffert werden, die das System nicht verlieren darf.

Weiter ist zu beachten, dass die Modellierung der monitorierten Größen in den HLSCs der internen Repräsentation des System-Interfaces entspricht. Daher

kann nicht vollständig von der Art der Input-Devices abstrahiert werden, bzw. Annahmen über deren Beschaffenheit müssen getroffen werden. Vergleichen Sie hierzu das folgende Beispiel 2.

Anweisung:

Erstellen Sie ein SC für jede Klasse monitorierter Größen aus dem Klassendiagramm, das Sie in Schritt 2 erstellt haben. Stellen Sie jeden möglichen Wert der monitorierten Größe als State dar. Aufschluss über mögliche Werte der Größen geben die Tabelle 5 und die Tabelle 7, die Sie in den vorbereitenden Schritten 3.1.4 und 3.1.5 erstellt haben.

Definieren Sie nun die möglichen Transitionen, die zu Zustandsübergängen zwischen den States führen. Hinweise auf Events, die einen Zustandsübergang der monitorierten Größen auslösen können, gibt das UC- Element „Beschreibung“. Zum Beispiel, alle Sätze deren Subjekt ein Akteur ist und deren Verb eine Aktivität beschreibt (z.B. starten, stoppen, aktivieren, eingeben, ändern, setzen, etc.), beschreiben mögliche Events, die den Wert einer monitorierten Größe beeinflussen könnten. Die Namen der Events sind frei wählbar, sollten aber intuitiv sein, um eine einfache Verfolgbarkeit zwischen den UCs und den HLSCs zu gewährleisten.

Machen Sie sich zu jedem identifizierten Event Gedanken darüber, ob das entsprechende inverse Event ebenfalls eine für das System sinnvolle Transition widerspiegelt.

Mögliche Optimierung:

Es gibt Inputs in das System, die ein bestimmtes Systemverhalten auslösen, aber nicht durch einen weiteren Input gestoppt werden. In einem solchen Fall ist es nicht notwendig einen Zustandsübergang der entsprechenden monitorierten Größe zu modellieren. Es genügt einen Event zu erzeugen der diese Eingabe repräsentiert. Machen Sie sich daher Gedanken, ob alle Zustände der monitorierten Größen unter Berücksichtigung dieses Gesichtspunktes sinnvoll sind. Beachten Sie hierzu insbesondere das folgende Beispiel 2.

Wie bereits in Kapitel 3.3.2 erwähnt, ist eine Festlegung zu treffen welche Daten von der monitorierten Größe aktiv gesendet werden und welche von den Klassen der UCs abgefragt werden müssen. Generell ist es dem Modellierer der HLSCs überlassen welche Realisierung gewählt wird. In diesen Guidelines wurden die monitorierten Größen als aktive Größen modelliert, d.h. auch Inputs die nicht wirklich aktiv sind werden aktiv an die UC-Klassen gesendet. Zum Beispiel wird das Erreichen der maximalen oder minimalen Endposition des Fensters aktiv von der monitorierten Variablen an die UC-Klassen gesendet. Dies erleichtert die Modellierung der HLSCs der UC-Klassen, da man dort das ständige Abfragen der Sensoren nicht modellieren muss. Alternativ könnte man diese

Daten auch passiv bereitstellen, d.h. die in den HLSCs der UC-Klassen muss diese Abfrage ständig durch einen „ev_Tick“ getriggert werden.

Rationale:

Sind die Werte der monitorierten Größe als States dargestellt, so kann der Leser sehr schnell und einfach eine Veränderung der Werte identifizieren. Die Events, die Zustandsübergänge auslösen, können direkt auf Aktivitäten, die in den UCs beschrieben sind, abgebildet werden.

Beispiel 1:

Im Funktionsblock „Fenstersteuerung“ der Beispielspezifikation wird die Variable „Kindersicherung“ verwendet. Nach Anwendung der vorbereitenden Schritte 3.1.4 und 3.1.5 ergeben sich folgende Informationen zu dieser monitorierten Größe:

Monitorierte Größe	Use Case	Wertebereich
Kindersicherung	Steuere Fond Fensterscheibe an	aktiv, nicht aktiv

Tabelle 11:

Beispiel-Variable Kindersicherung

Diese Informationen werden nun in das SC der Klasse „M_Kindersicherung“ abgebildet, welche diese beobachtete Größe modelliert. Abbildung 6 zeigt das SC der Klasse „M_Kindersicherung“. Die States „s_Nicht_Aktiv“ und „s_Aktiv“ ergeben sich direkt aus den beiden möglichen Werten der Variable und repräsentieren den aktuellen Wert der monitorierten Größe.

Es ist denkbar, dass ein Benutzer des Systems die Kindersicherung über ein Interface aktivieren bzw. deaktivieren kann. Eine solche Aktion würde in der UC Beschreibung als Anforderung formuliert, z.B. „Der Fahrer aktiviert die Kindersicherung“. Dieser Satz gibt Hinweise auf mögliche Transitionen, die zu Zustandsübergängen der monitorierten Größe „Kindersicherung“ führen. Aus diesem Satz kann man direkt das Event „ev_Aktiviere_KSicherheit“ ableiten, das zu einem Zustandsübergang von „s_Nicht_Aktiv“ nach „s_Aktiv“ führt.

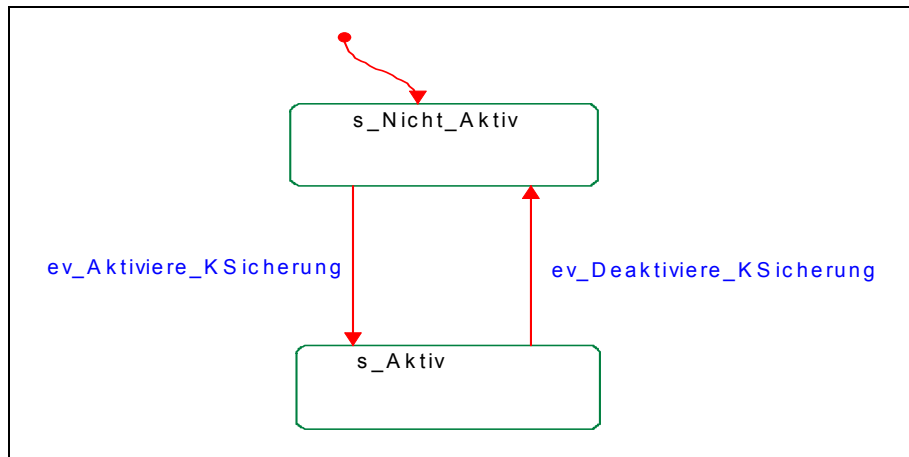


Abbildung 6: SC der monitorierten Größe „M_Kindersicherung“ mit zwei States

Beispiel 2:

Eine weitere monitorierte Größe aus dem Funktionsblock „Fenstersteuerung“ der Beispielspezifikation ist der „Fahrer_Input“. Nach Anwendung der vorbereitenden Schritte 3.1.4 und 3.1.5 ergeben sich folgende Informationen zu dieser monitorierten Größe:

Monitorierte Größe	Use Case	Wertebereich
Fahrer_Input	Steuere Andere Fenster-scheibe an	stop, partiell hoch, partiell runter, total hoch, total runter

Tabelle 12: Beispiel-Variable "Fahrer_Input"

Die UC-Beschreibung sieht verschiedene Arten von Bewegungen vor (partielle/totale Ansteuerung) sowie verschiedene Richtungen, in die das Fenster bewegt werden kann (hoch/runter). Durch die verschiedenen Möglichkeiten der Ansteuerung werden Annahmen über die Beschaffenheit der Input-Devices in das Modell übertragen. Bei einer totalen Ansteuerung betätigt der Benutzer nur einmal sein Interface und das entsprechende Fenster wird automatisch in die obere oder untere Endposition gefahren, d.h. es wird in diesem Fall ein kontinuierliches Signal erzeugt. Im Falle einer partiellen Ansteuerung, muss der Benutzer sein Interface einmal betätigen, um die entsprechende Bewegung zu starten und ein zweites mal, um die Bewegung wieder zu stoppen. Diese Bewegungsarten werden im Modell durch die Wertebereiche der monitorierten Größe repräsentiert.

Abbildung 7 zeigt das SC der Klasse „M_Fahrer_Input“, die der monitorierten Größe „Fahrer_Input“ entspricht. Diese Abbildung repräsentiert bereits die optimierte Variante des SC. Der Guideline folgend müssten die States „s_Total_Hoch“ und „s_Total_Runter“ im SC vorhanden sein, damit alle möglichen Werte als State repräsentiert sind.

Die monitorierte Größe würde diese States annehmen, wenn der Benutzer eine totale Ansteuerung des Fensters betätigt („Total_Runter“ oder „Total_Hoch“). Diese Ansteuerung muss durch keinen weiteren Event gestoppt werden. Daher würde die monitorierte Größe in den entsprechenden State „s_Total_Hoch“ oder „s_Total_Runter“ wechseln, diesen Zustand aber sofort wieder verlassen, da das System die Ansteuerung des Fensters automatisch übernimmt. Daher ist es nicht nötig diese möglichen Werte als States zu repräsentieren. Es genügt die entsprechenden Events („ev_Start_Total_Runter“ und „ev_Start_Total_Hoch“) zu modellieren.

Das Event „ev_Stop_Input“ repräsentiert, dass der Benutzer bei einer partiellen Ansteuerung diese beendet. Das Beenden der Ansteuerung wird also ebenfalls als Event modelliert. Die Operationen „o_Send_Partiiell_x“ bzw. „o_Send_Total_x“ leiten die Art der Ansteuerung an die UC-Klassen „UC_Stelle_Eigene_Ein“ und „UC_Stelle_Andere_Ein“ weiter (vgl. Sie zur Verwendung von Operationen auch Kapitel 3.8). Die Herleitung der übrigen States und Events entspricht dem Vorgehen, wie es in der Anweisung beschrieben wurde.

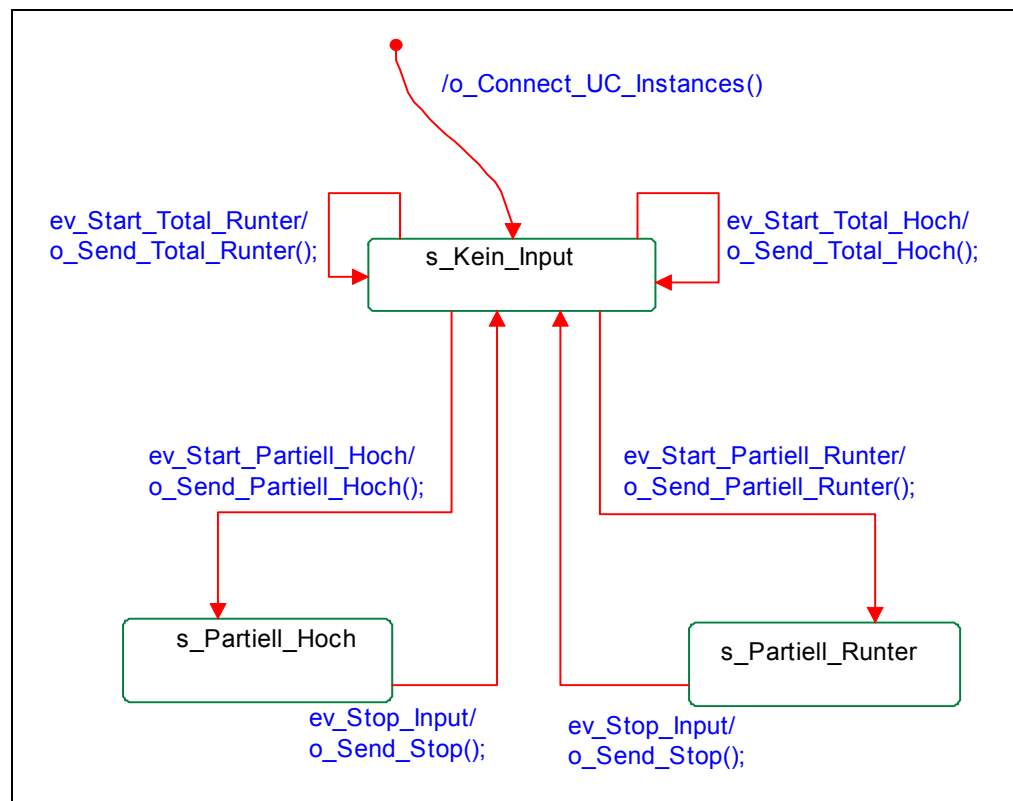


Abbildung 7: SC der Klasse "M_Fahrer_Input"

3.6 Schritt 5: Modellieren der SC der kontrollierten Größen

Auch die Modellierung der kontrollierten Größen entspricht einer internen Repräsentation des System-Interface. Bei der Modellierung dieser Größen muss man daher darauf achten, dass die Ansteuerung der realen kontrollierten Größen über das Interface korrekt abgebildet wird. Falls nötig, muss z.B. ein kontinuierlicher Output in der kontrollierten Größe erzeugt werden, der die kontinuierliche Ansteuerung der realen kontrollierten Größen repräsentiert.

Anweisung:

Die konkrete Modellierung der kontrollierten Größen läuft analog zur Modellierung der monitorierten Größen (siehe Schritt 4, Kapitel 3.5).

Der Einzige Unterschied besteht in der Ableitung der Events. Im Falle von kontrollierten Größen geben Sätze aus der Beschreibung der UCs Hinweise auf mögliche Zustandsübergänge, deren Subjekt das modellierte System selbst ist, oder ein Teilsystem, das innerhalb des modellierten Systems liegt. D.H. hier ist der Akteur nicht aus der Umgebung des modellierten Systems, sondern das System oder ein Teilsystem ist hier der „Akteur“.

Rationale:

Sind die Werte der monitorierten Größe als States dargestellt, so kann der Leser sehr schnell und einfach eine Veränderung der Werte identifizieren. Die Events, die Zustandsübergänge auslösen, können direkt auf Aktivitäten, die in den UCs beschrieben sind, abgebildet werden.

Beispiel 1:

Im Funktionsblock „Fenstersteuerung“ der Beispielspezifikation wird die Variable „Aktuelle Fensterposition“ verwendet. Nach Anwendung der vorbereitenden Schritte 3.1.4 und 3.1.5 ergeben sich folgende Informationen zu dieser kontrollierten Größe:

Monitorierte Größe	Use Case	Wertebereich
Aktuelle Fensterscheibenbewegung	Steuere Andere Fensterscheibe an	stop, Bewegung hoch, Bewegung runter

Tabelle 13: Beispiel-Variablen "Aktuelle Fensterscheibenbewegung"

Diese Informationen werden nun in das SC der Klasse „C_Fenster_Position_Soll“ abgebildet, welche die kontrollierte Größe „aktuelle Fensterscheibenbewegung“ repräsentiert. Das System versetzt die kontrollierte Größe in den Zustand „s_Bewege_Hoch“ oder „s_Bewege_Runter“, basierend auf den Eingaben aus der Umgebung. Die Zustände ergeben sich direkt aus dem Wertebereich der Größe. Die Events ergeben sich aus der UC Beschreibung, z.B. aus Sätzen wie „Der Benutzer betätigt Taster A. Das System bewegt das Fenster nach oben“.

Die in der Abbildung beschriebenen Aktionen „a_Position++“ und „a_Position--“ modellieren das Reduzieren bzw. das Erhöhen des Wertes, der die Fenster-scheibenposition repräsentiert. Der Wert wird bei jedem Event „ev_Tick“ erhöht bzw. erniedrigt, solange der entsprechende Zustand angenommen wird. Für die Ableitung solcher Aktionen aus den UCs gibt es keine speziellen Richtlinien. Solche Aktionen müssen bei Bedarf eingesetzt werden. Im konkreten Beispiel hätte man die Aktionen auch mit Operationen realisieren können (siehe dazu Kapitel 3.8).

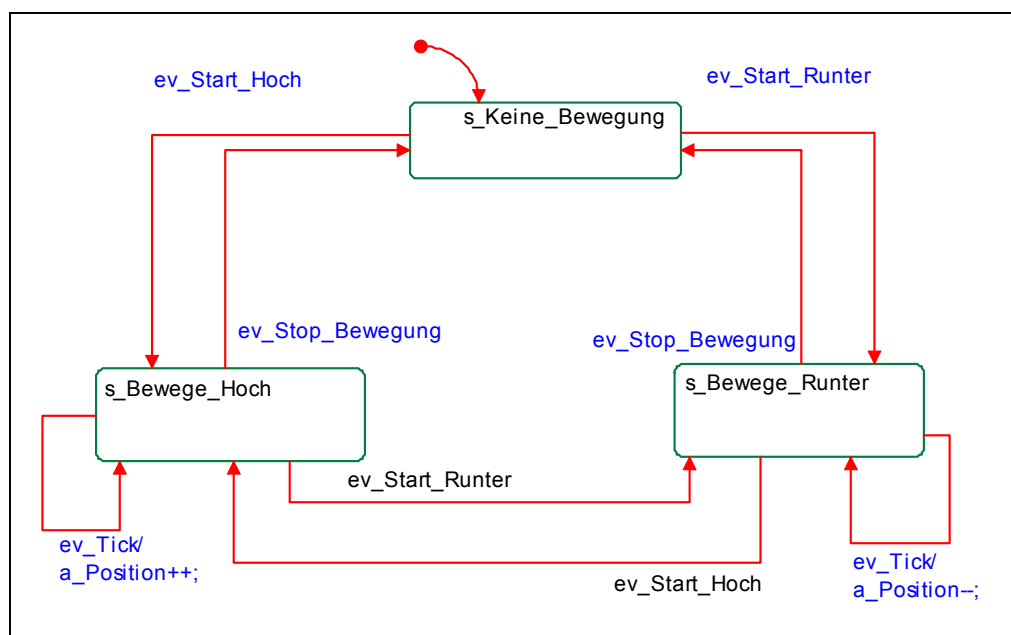


Abbildung 8: SC der kontrollierten Größe "C_Fensterposition_Soll"

3.7 Schritt 6: Erstellen Sie die Use Case Statecharts (UCSCs)

In diesem Schritt werden die SC der Klassen modelliert, die einen UC repräsentieren. Neben der Modellierung der States und Transitionen, die sich aus der UC-Beschreibung ergeben, werden hier auch die in den UCs beschriebenen Ausnahmefälle und deren Realisierung in den HLSCs behandelt.

3.7.1 Erstellung der HLSCs für die UC-Klassen

Anweisung:

Modellieren Sie ein SC für jede Klasse aus dem Klassendiagramm, die einen UC repräsentiert. Modellieren Sie dazu zunächst die folgenden States:

- Einen State des Typs „Idle“ und
- Einen State des Typs „System Reaction“

Modellieren Sie nun eine Transition vom Zustand des Typs „Idle“ in den Zustand vom Typ „System Reaction“. Diese Transition entspricht dem ersten Schritt in der UC- Beschreibung in dem der UC „aktiv“ wird. Definieren Sie den Event, durch den diese Transition getriggert wird.

Modellieren Sie eine Transition vom Zustand des Typs „System Reaction“ in den Zustand vom Typ „Idle“. Diese Transition repräsentiert das Beenden des UC, d.h. den letzten Schritt der UC-Beschreibung. Benennen Sie den Event dieser Transition so, dass deutlich wird, dass die System- Reaktion, die im UC beschrieben wird, beendet ist. Hinweise, welche Events diese Transition auslösen, geben Ihnen die Elemente „Beschreibung“ und „Nachbedingungen“ der textuellen UC- Beschreibung, sowie UC- Wechsel. Wird ein UC auf verschiedene Arten und Weisen beendet, dann modellieren Sie für jede Möglichkeit eine entsprechende Transition.

Modellieren Sie zyklische Transitionen in den Zustand vom Typ „System Reaction“, um sich wiederholende Systemfunktionalitäten zu beschreiben, die in diesem Zustand realisiert werden. (Beachten Sie hierzu auch Kapitel 3.8, in dem die Verwendung von Operationen und Verfeinerungen von SC beschrieben wird.)

Rationale:

Durch diese einfache Struktur der SC auf dieser Ebene wird eine sehr gute Übersichtlichkeit der SC erzielt. Weiter wird eine sehr einfache Verfolgbarkeit zu den UCs erreicht. Das Verhalten eines UC, bzw. einer Klasse, die einen UC repräsentiert, kann stets durch die beiden States vom Typ „Idle“ und „System Reaction“ beschrieben werden.

Beispiel:

Die folgende Abbildung zeigt das Statechart der Klasse „UC_Stelle_Andere_Ein“, welche den UC „Steuere andere Fensterscheibe an“ aus der Beispielspezifikation repräsentiert. Auf oberster Ebene existieren lediglich die Zustände vom Typ „Idle“ und „System Reaction“. Ein Wechsel von „s_Idle“ nach „s_System_Reaction“ findet dann statt, wenn das System einen entsprechenden Input von den monitorierten Größen erhält („ev_Eingabe_Start“). Dieses Event wird in dem SC der Klasse „M_Fahrer_Input“ erzeugt, sobald eine Eingabe aus der Umgebung empfangen wird. Dies entspricht dem ersten Schritt des Elementes „Beschreibung“ der textuellen UC- Beschreibung. Entsprechend dem Parameter „p_Typ“ des Events „ev_Eingabe_Start“, wird entweder der UC „UC_Stelle_Total_Ein“ oder der UC „UC_Stelle_Partiiell_Ein“ angesteuert (über die Operation „o_Starte_UC“, die den entsprechenden UC mit der Bewegungsrichtung versorgt). Die angesteuerten UCs repräsentieren die vom UC „Steuere andere Fensterscheibe an“ inklu-

dierten UCs „Steuere Fensterscheibe Partiiell an“ und „Steuere Fensterscheibe Total an“

Die zyklische Transition, die durch das Event „ev_Eingabe_Start“ ausgelöst wird, modelliert die Situation, dass ein Benutzer während der Systemreaktion eine neue Eingabe tätigt. So ist z.B. möglich, dass der Fahrer zunächst eine totale Ansteuerung nach oben vornimmt, diese dann aber unterbricht durch eine totale Ansteuerung nach unten. In einem solchen Fall muss die System Reaktion nicht abgebrochen werden, aber die neuen Parameter an die UCs übertragen werden. Dies wird durch die zyklische Transition realisiert, da die neue Richtung und der neue Bewegungstyp, über die entsprechenden Parametern, an die UCs weitergeleitet werden.

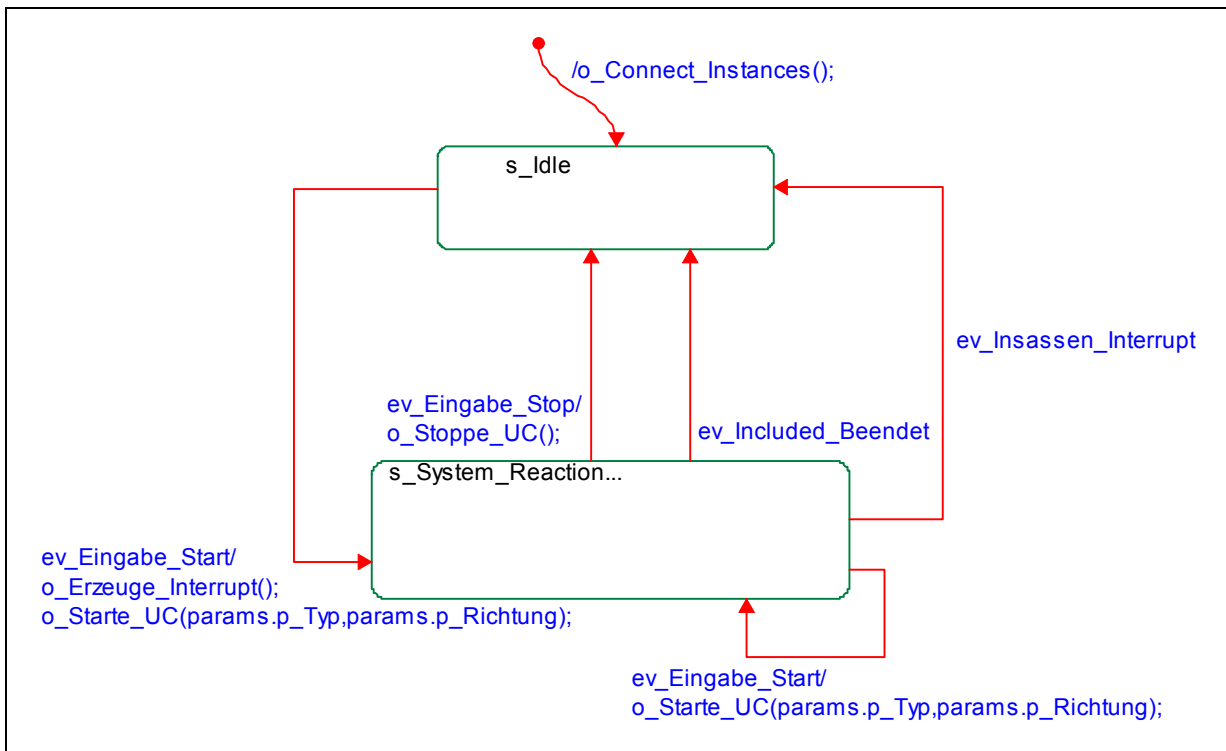


Abbildung 9 Statechart der Klasse „UC_Stelle_Andere_Ein“

Die aus dem Zustand „s_System_Reaction“ zum Zustand „s_Idle“ führenden Transitionen repräsentieren die in der UC Beschreibung aufgeführten Möglichkeiten den UC „Steuere andere Fensterscheibe an“ zu beenden. Das Event „ev_Eingabe_Stop“ modelliert, dass keine weitere Eingabe aus der Umgebung mehr empfangen wird. Das Event „ev_Included_Beendet“ zeigt an, dass einer der inkludierten UC abgeschlossen wurde und nun auch der inkludierende UC beendet wird, d.h. die Nachbedingung erfüllt ist.

3.7.2 Modellierung der Ausnahmefälle der UC

Neben den in 3.7.1 beschriebenen Situationen, die zu Zustandsübergängen führen, werden die Transitionen der SC auch durch Ausnahmefälle in den UCs beeinflusst. Wir unterscheiden zwischen drei Arten von Ausnahmefällen:

- Ausnahmefälle, die zu anderen UCs führen.
- Ausnahmefälle, die die Durchführung eines UC (-Schrittes) verhindern.
- Ausnahmefälle, die zum Abbruch der Systemreaktion führen.

Fall 1: Ausnahmefälle, die zu anderen Use Cases führen

Anweisung:

Modellieren Sie Ausnahmefälle, die einen neuen UC starten und den aktuell aktiven UC unterbrechen, als Transition zwischen den Zuständen vom Typ „System Reaction“ und denen vom Typ „Idle“. Hinweise auf solche UC- Wechsel gibt Ihnen die Tabelle, die Sie im vorbereitenden Schritt 3.1.2 erstellt haben.

Modellieren Sie eine Transition vom Zustand des Typs „System Reaction“ in den Zustand vom Typ „Idle“ im SC des UC, der unterbrochen wird. Benennen Sie das Event, das diese Transition triggert so, dass deutlich wird, dass es sich um eine Unterbrechung durch einen anderen UC handelt.

Fügen Sie im SC des UC, der einen anderen UC unterbricht, an der Transition vom Zustand des Typs „Idle“ in den Zustand des Typs „System Reaction“ eine Aktion (Operation) ein. Diese Aktion (Operation) muss dann das Event aufrufen, das in dem SC des unterbrochenen UC zum Zustandsübergang vom Zustand des Typs „System Reaction“ in den Zustand vom Typ „Idle“ führt.

Rationale:

Wenn sich UCs gegenseitig unterbrechen, dann muss sichergestellt werden, dass in einem solchen Fall nur einer der UCs aktiv ist, d.h. nur einer der UCs im Zustand vom Typs „System Reaction“ ist. Weiter wird so sichergestellt, dass Ausnahmen mit sehr geringem Aufwand in den SCs der UCs erkennbar sind und daher eine sehr gute Verfolgbarkeit zu den UCs selbst sicher gestellt wird.

Beispiel:

Die folgende Abbildung zeigt die Realisierung des Ausnahmefalls, dass in einem UC, hier der UC „UC_Stelle_Total_Ein“, ein anderer UC, hier „UC_Stelle_Partiiell_Ein“ aufgerufen wird. Die Abbildung zeigt auf der linken Seite einen Ausschnitt aus dem SC der Klasse „UC_Stelle_Partiiell_Ein“ und auf der rechten Seite ein Teil des SC der Klasse „UC_Stelle_Total_Ein“. In einem solchen Ausnahmefall muss der aktuell aktive UC, im Beispiel also der UC „UC_Stelle_Total_Ein“ in den Zustand „s_Idle“ vom Typ „Idle“ versetzt werden.

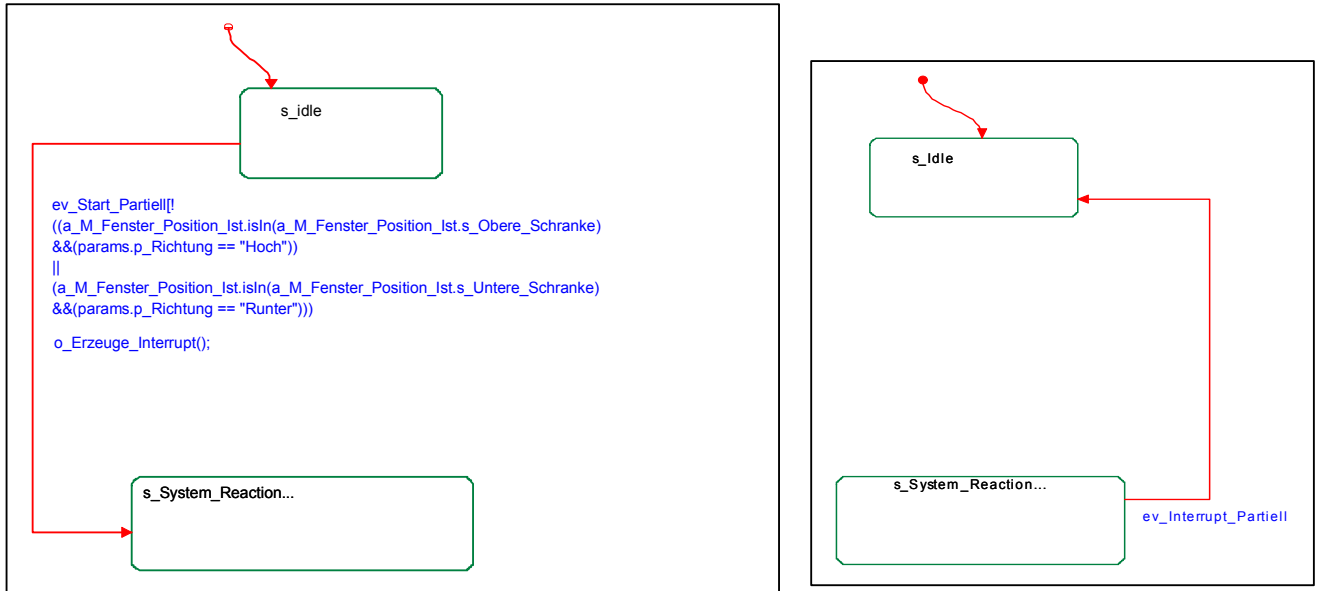


Abbildung 10: Modellieren von Ausnahmefällen

Dies wird durch das Event „ev_Interrupt_Partiiel“ realisiert. Dieses Event wird am Übergang vom Zustand „s_Idle“ in den Zustand „s_System_Reaction“ des SC der Klasse „UC_Stelle_Partiiel_Ein“ in der Operation „o_Erzeuge_Interrupt()“ generiert.

Fall 2: Ausnahmefälle, die die Durchführung eines UC (-Schrittes) verhindern

Anweisung:

Modellieren Sie diese Art von Ausnahmefälle als Guards an den Zustandsübergängen zwischen den States vom Typ „Idle“ zu dem State vom Typ „System Reaction“. Erweitern Sie dazu die entsprechende Transition zwischen diesen Zuständen um einen Guard, der das Vorhandensein einer entsprechenden Ausnahmesituation überprüft. Hinweise auf solche Ausnahmesituationen geben Ihnen die Ausnahmefälle der UCs, bei deren Eintreten das System nicht reagiert (die erwartete Systemreaktion lautet „Keine Reaktion“).

Rationale:

Die Modellierung von solchen Ausnahmen als Guards ist die einzige Möglichkeit zu modellieren, dass in bestimmten Fällen keine Systemreaktion erfolgen darf. Weiter wird durch diese Art der Modellierung sichergestellt, dass Ausnahmen mit sehr geringem Aufwand in den SCs der UC- Klassen erkennbar sind und daher eine sehr gute Verfolgbarkeit zu den UCs erreicht werden kann. Außerdem ist.

Beispiel:

Betrachten Sie erneut Abbildung 10. Das Beispiel zeigt, dass die Transition vom Zustand „s_Idle“ in den Zustand „s_System_Reaction“ bei dem linken SC- Ausschnitt durch einen Guard geschützt ist. Dieser Guard repräsentiert die im UC „Steuere Fensterscheibe Total an“ beschriebene Ausnahme, dass sich das angesteuerte Fenster in der oberen oder unteren Endposition befindet und das Fenster nach oben bzw. unten bewegt werden soll. In einem solchen Fall soll das System nicht reagieren. Dies wird dadurch erreicht, dass der Zustandsübergang von „s_Idle“ nach „s_System_Reaction“ durch den Guard verhindert wird, sobald diese Bedingung gültig ist.

Fall 3: Ausnahmefälle, die zum Abbruch der Systemreaktion führen**Anweisung:**

Modellieren Sie Ausnahmen dieser Art als Transitionen vom Zustand des Typs „System Reaction“ zu dem Zustand vom Typ „Idle“. Fügen Sie dazu eine neue Transition in das SC der Klasse ein, die den UC beschreibt, in dem diese Ausnahme auftreten kann.

Ergänzen Sie die Transition um einen Event, der das Eintreten der Ausnahme modelliert. Hinweise auf solche Ausnahmesituationen geben Ihnen die Ausnahmefälle der UCs, bei deren Eintreten das System eine bestimmte Aktivität abbrechen muss. Das Event, das die Transition auslöst, wird entweder in einer monitorierten Variablen oder im SC der UC- Klasse selbst erzeugt.

Ergänzen Sie die Transition um eine Aktion, die sicherstellt, dass das Eintreten der Ausnahme an die kontrollierten Größen propagiert wird.

Rationale:

Das Verlassen des Zustandes vom Typ „System Reaction“ verdeutlicht sehr einfach die Tatsache, dass eine bestimmte Systemreaktion durch diese Ausnahme abgebrochen werden muss. Dadurch wird sichergestellt, dass Ausnahmen mit sehr geringem Aufwand in den SC der UC- Klassen erkennbar sind und daher eine sehr gute Verfolgbarkeit zu den UCs gewährleistet ist.

Beispiel:

Diese Art der Ausnahme ist in Abbildung 11 modelliert. Die Abbildung repräsentiert einen Ausschnitt aus dem SC der Klasse, die den UC „Steuere Fensterscheibe partiell an“ modelliert. In diesem UC ist folgende Ausnahme spezifiziert: Wird während der Ansteuerung des Fensters, die obere oder untere Endposition des Fensters erreicht, dann soll das System die Reaktion, d.h. die Fensterbewegung abbrechen.

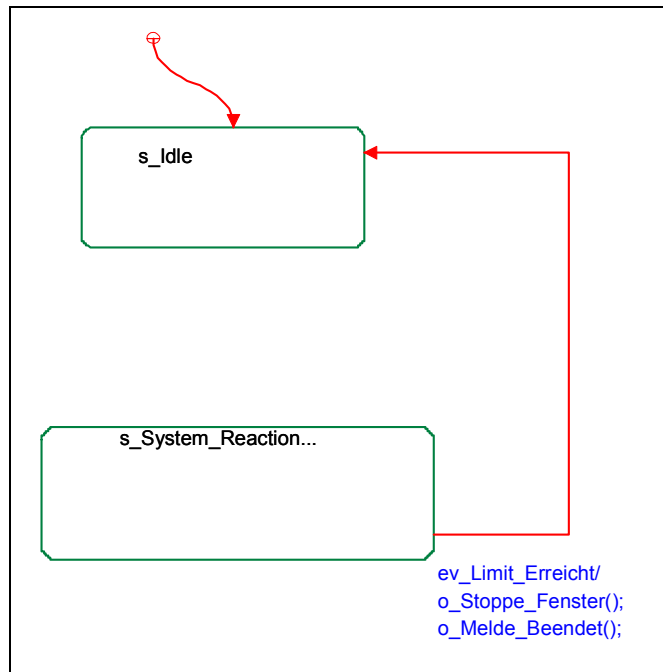


Abbildung 11: Realisierung einer Ausnahme die zum Abbruch einer Reaktion führt

Dieser Ausnahmefall ist als Transition vom Zustand „s_System_Reaction“ in den Zustand „s_Idle“ modelliert. Das Event „ev_Limit_Erreicht“ wird in der monitorierten Größe „M_Fenster_Position_ist“ erzeugt, genau dann, wenn die aktuelle Ist-Position des Fensters gleich der oberen bzw. unteren Schranke ist. Wird dieses Event empfangen, dann ist die momentan durchgeführte Bewegung abzubrechen, d.h. der Zustand „s_System_Reaction“ muss verlassen werden. Durch die Operation „o_Stoppe_Fenster“ wird das Eintreten der Ausnahme an die kontrollierten Größen propagiert.

3.8 Schritt 7: Verfeinern der Systemreaktion (insbesondere Umsetzung von Regeln)

Die Verfeinerung der Systemreaktion geschieht auf der Grundlage der in der textuellen UC- Beschreibung definierten „Regeln“ und „Beschreibungen“. Zur Verfeinerung der Systemreaktionen ergeben sich die folgenden beiden Fälle. Beachten Sie, dass die beiden Fälle zur Realisierung dieses Schrittes sehr stark zusammenhängen und daher gemeinsam betrachtet werden sollten.

Fall 1: Die Systemreaktion muss schnell visuell erkennbar sein.

Optionen:

- Option 1: Modellieren der Systemreaktionen als Verfeinerung des States „System_Reaction“.
- Option 2: Modellieren der Systemreaktionen als Operation in den SCs.

	VerAbb	VerErg	ModAuf	WarAuf
Opt.1	0	+	-	0
Opt.2	0	0	+	0

Tabelle 14: Option/Kriterien Matrix zur Modellierung von Regeln

Entscheidung: Modellieren der Systemreaktionen als Verfeinerung des States (Option 1)

Vorteile:

Die modellierten Regeln und verfeinerten Systemreaktionen sind auf den ersten Blick lesbar (VerErg+). Dadurch wird die einfache Verfolgbarkeit zwischen den UC und den HLSC realisiert (VerAbb+). Weiter kann das Systemverhalten, das nach außen sichtbar sein muss, simuliert werden. Dadurch ist es möglich die korrekte Umsetzung von wesentlichen Regeln (Systemreaktionen) bereits auf diesem hohen Abstraktionsniveau zu überprüfen. Dies ist positiv im Hinblick auf die Qualitätssicherung zu werten.

Nachteile:

Wenn die Regeln und die zu verfeinernden Systemreaktionen sehr komplex sind, dann kann die Modellierung dieser Aspekte als SC zu einer Zustandsexplosion führen, d.h. unter Umständen werden sehr viele States nötig, um die Regeln und verfeinerten Reaktionen zu beschreiben. Dies führt dann zu einem höheren Modellierungsaufwand (ModAuf-) und zu einer reduzierten Verständlichkeit der Abbildung zwischen den UCs und den HLSCs (VerAbb-).

Abgelehnte Option:

Das wesentliche Problem bei der Realisierung von Regeln bzw. Verfeinerungen der Systemreaktionen als Operationen ist, dass es nicht möglich ist, das so beschriebene Verhalten zu visualisieren. Zum Beispiel wären Interaktionen zwischen Klassen in solchen Operationen verborgen und somit während einer Simulation nicht sichtbar, also nicht direkt überprüfbar.

Anweisung:

Ist die Visualisierbarkeit einer Regel bzw. einer verfeinerten Systemreaktion essentiell, dann modellieren Sie diese als Verfeinerung des entsprechenden States vom Typ „System Reaction“. Kann eine Regel unterschieden werden in einen Wahrnehmungsanteil (Dinge, die vom System wahrgenommen werden) und einen Kontrollanteil (vom System kontrollierte Dinge), dann unterscheiden Sie die folgenden zwei Fälle:

- 1.) Modellieren Sie den Teil einer Regel, der die Dinge beschreibt, die vom System kontrolliert werden als Verfeinerung des State vom Typ „System Reaction“
- 2.) Modellieren Sie den Teil einer Regel, der Dinge beschreibt, die vom System wahrgenommen werden, als parallelen State zu der in 1. eingefügten Verfeinerung.

Rationale:

In einer Situation, in der eine Regel oder Systemreaktion unbedingt visualisiert und simuliert werden muss, schließt sich die Option 2 von selbst aus. Allerdings gibt es Fälle in denen die Visualisierbarkeit nicht zwingend notwendig ist. Dann muss eine Abwägung zwischen den beiden Optionen getroffen werden (beachten Sie dazu auch Fall 2 dieses Schrittes). Man erkennt an der Option/Kriterien Matrix (Tabelle 14), dass es hier wirklich auf den Kontext ankommt, da man streng genommen Option 2 wählen müsste, da diese Option insgesamt positiver bewertet wurde.

Beispiel:

Das folgende Beispiel erläutert die Realisierung einer Regel aus den UCs durch die Verfeinerung des States vom Typ „System Reaction“. Der UC „Steuere Fensterscheibe Partiiell an“ aus dem Funktionsblock „Fenstersteuerung“ der Beispielspezifikation enthält die Regel, dass, sobald eine Blockierung des Fensters auftritt, das System das entsprechende Fenster in die untere Endposition bewegen soll. Abbildung 12 zeigt einen Ausschnitt der Verfeinerung des States „s_System Reaction“ der Klasse „UC_Stelle_Partiiell_Ein“, die den UC „Steuere Fensterscheibe Partiiell an“ repräsentiert. Diese Verfeinerung realisiert die Regel des UC wie folgt:

Der obere Teil der Abbildung beschreibt den Teil der Regel, der den Kontrollanteil widerspiegelt, d.h. hier wird die verfeinerte Reaktion des Systems beschrieben. Der Zustandsübergang von „s_System_Reaction“ in den State „s_Einklemmschutz“ wird getriggert, wenn das System eine Blockierung des Fensters registriert. Dies ist der Fall, wenn die Ist-Position des Fensters nicht mehr mit der Soll-Position des Fensters übereinstimmt, was in dem Guard der Transition vom Zustand „s_Fenster_Nicht_Blockiert“ in den Zustand „s_Fenster_Blockiert“ im unteren Teil der Abbildung überprüft wird. Der untere Teil beschreibt somit den Teil der Regel, der die Aspekte beschreibt, die vom System wahrgenommen werden. Die Überprüfung selbst, ob eine Blockierung vorliegt wird in der Operation „o_Pruefe_Blockierung“ realisiert. Befindet sich das System dann im Zustand „s_Einklemmschutz“, wird das blockierte Fenster in die untere Endposition bewegt. Die Ansteuerung des Fensters ist dabei in der Operation „o_Starte_Fenster“ gekapselt.

Durch diese Art der Modellierung ist die Realisierung der Regel sofort sichtbar und deren Umsetzung kann in einer Simulation überprüft werden.

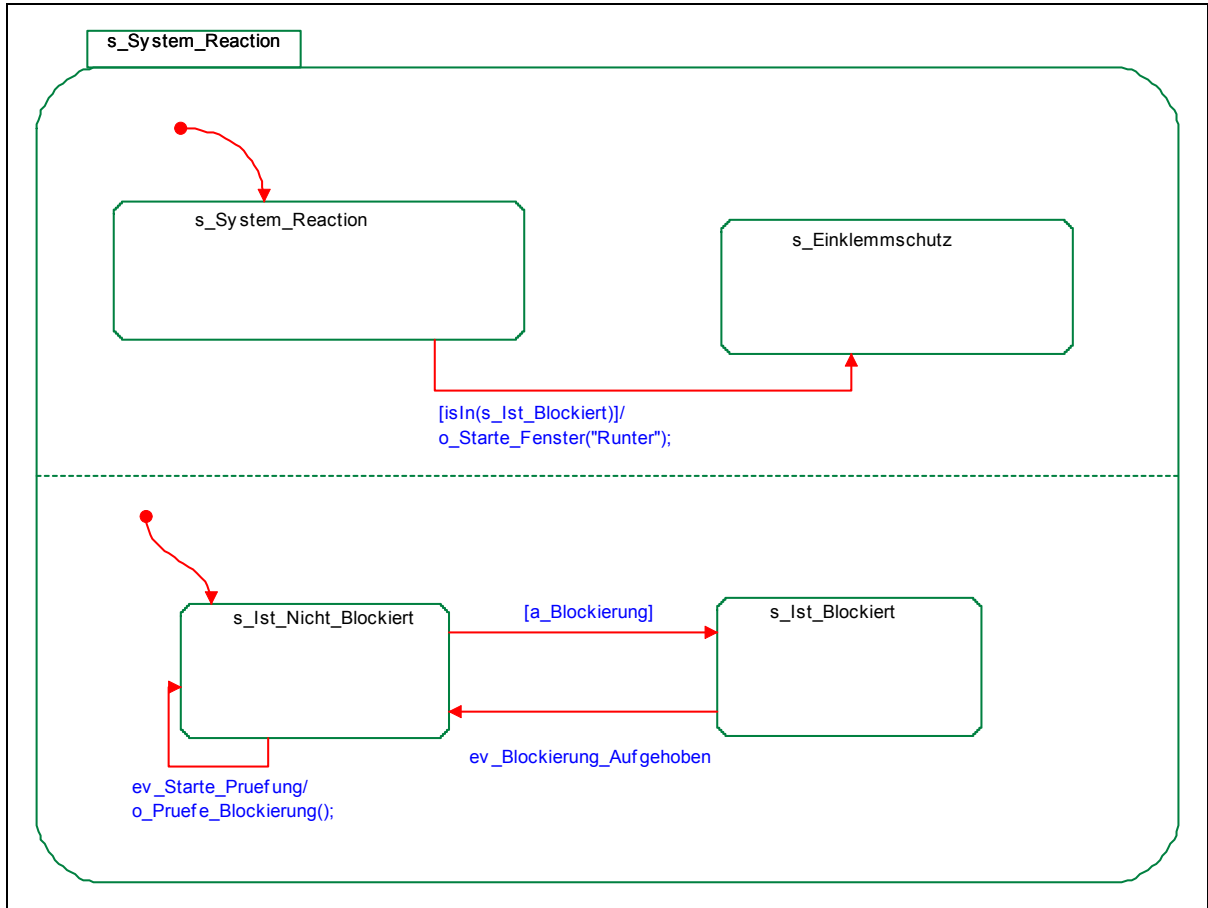


Abbildung 12: Substatechart des UC "UC_Stelle_Partiiell_Ein"

Fall 2: Realisierung der Regel bzw. der Systemreaktion muss nicht visualisierbar sein

Optionen:

- Option 1: Modellieren der Systemreaktionen als Verfeinerung des States „System Reaction“
- Option 2: Modellieren der Systemreaktionen als Operation in den SCs.

	VerAbb	VerErg	ModAuf	WarAuf
Opt.1	0	0	-	0
Opt.2	0	0	+	0

Tabelle 15: Option/Kriterien Matrix Regel nicht visualisieren

Entscheidung: Modellieren der Systemreaktionen als Operation (Option 2)

Vergleichen Sie hierzu auch die Diskussion zu Fall 1 dieses Schrittes.

Vorteile:

Durch die Modellierung als Operation ist es möglich immer wiederkehrende Operationen zu kapseln und per einschlägigem Namen zu verstecken (ModAuf+). Weiter ist es möglich durch die Operationen komplexe Transitionen einfacher und lesbarer zu beschreiben (VerErg+).

Nachteile:

Das wesentliche Problem bei der Realisierung von verfeinerten Systemreaktionen als Operationen ist, dass es nicht möglich ist, das so beschriebene Verhalten zu visualisieren, was zu negativen Auswirkungen auf qualitätssichernde Maßnahmen führt. Weiter ist die Abbildung zwischen den UCs und den HLSCs nicht sehr intuitiv (VerErg-)

Abgelehnte Option:

Die Realisierung von Regeln und verfeinerten Systemreaktionen als SC kann zu sehr komplexen Transitionen und Zustandsexplosionen führen, was zu einer Verringerung der Lesbarkeit der SC führt (VerErg-, ModAuf-). Allerdings sind so modellierte Regeln sehr einfach für den Leser der HLSCs sichtbar (VerAbb+, VerErg+).

Anweisung:

Wenn das Modellieren als Statechart zu umfangreich wird, und die Erstellung von „Hilfsstates“ erforderlich wird, dann modellieren Sie die komplexen Systemreaktionen als Operation.

Verwenden Sie Operationen insbesondere dann, wenn diese Funktionen beschreiben, die für den Benutzer nicht sichtbar sind, bzw. kein Systemverhalten beschreiben, z.B. Verknüpfung von Instanzen der Klassen (beachten Sie dazu auch Fall 1 dieses Schrittes).

Beschreiben Sie die Operationen in Rhapsody in J mittels Java Code in den dafür vorgesehenen Editoren. Betrachten Sie dazu das nachfolgende Beispiel!

Rationale:

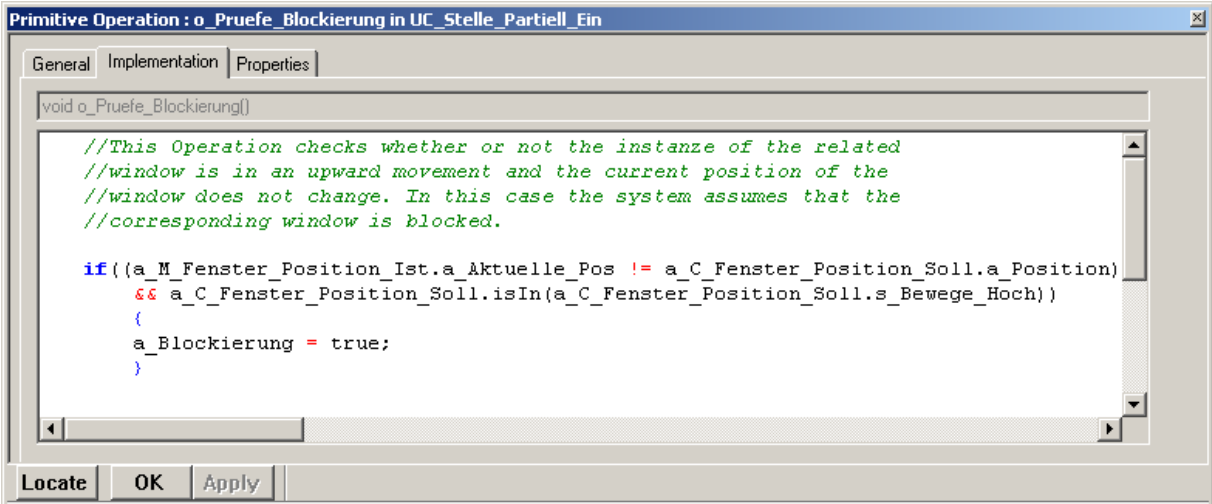
Ist die Visualisierbarkeit und die Simulierbarkeit bestimmter Systemfunktionalitäten nicht zwingend notwendig, dann muss der Modellierer eine Abwägung treffen, ob es sinnvoll ist die Funktionalität mittels States oder durch Operationen zu realisieren. Im Falle von internen Systemfunktionen und Funktionalitäten, die kein nach außen sichtbares Verhalten beschreiben, schließt sich Option 1 von selbst aus, da in diesem Fall der Leser der SC nur verwirrt werden würde und die SC überflüssige Zustände und Transitionen beinhalten würden.

Beispiel:

Zum Verdeutlichen der Modellierung des Systemverhaltens durch Operationen, betrachten wir nun erneut den UC „Steuere Fensterscheibe Partiiell an“ aus dem Funktionsblock „Fenstersteuerung“ der Beispielspezifikation.

Betrachten Sie erneut Abbildung 12, die einen Ausschnitt des SC der Klasse „UC_Stelle_Partiiell_Ein“ zeigt. Beim Auftreten des Events „ev_Starte_Pruefung“ wird die Operation „o_Pruefe_Blockierung“ aufgerufen. Diese Operation überprüft ob eine Blockierung des Fensters vorliegt, d.h. ob eine Bewegung des Fensters nach oben durchgeführt wird, die Fensterscheibenposition sich aber nicht verändert. Wie das System feststellt, ob ein Fenster blockiert ist, ist für den Benutzer des Systems nicht von Interesse und muss nicht visualisiert werden.

Abbildung 13 zeigt die Implementierung der Operation „o_Pruefe_Blockierung“.



```
void o_Pruefe_Blockierung()
//This Operation checks whether or not the instanze of the related
//window is in an upward movement and the current position of the
//window does not change. In this case the system assumes that the
//corresponding window is blocked.

if((a_M_Fenster_Position_Ist.a_Aktuelle_Pos != a_C_Fenster_Position_Soll.a_Position)
&& a_C_Fenster_Position_Soll.isIn(a_C_Fenster_Position_Soll.s_Bewege_Hoch))
{
    a_Blockierung = true;
}
```

Abbildung 13: Operationen im Code Editor

4 Zusammenfassung und Ausblick

In diesem Bericht wurde aufgezeigt, wie man systematisch Use Cases in High Level Statecharts überführen kann. Dazu wurden Richtlinien beschrieben, die diesen schrittweisen Ableitungsprozess definieren. Weiter wurden alle Aktivitäten, die in den Richtlinien durchgeführt werden und alle Entscheidungen bzgl. Modellierungsoptionen durch Rationales begründet. Dadurch werden die Richtlinien leichter änderbar und sind flexibler anwendbar, da den Benutzern der Richtlinien alternative Modellierungen aufgezeigt werden.

Eine Vorgängerversion dieser Richtlinien wurde in einem Studenten Seminar an der Technischen Universität Kaiserslautern angewendet. Die Studenten konnten die Richtlinien problemlos anwenden und bewerteten die Richtlinien als sehr hilfreich. Insbesondere die gute Verfolgbarkeit zwischen den Use Cases und den High Level Statecharts wurde als sehr positiv bewertet. Die Erfahrungen der Studenten wurden genutzt, um die Richtlinien weiter zu verfeinern und zu optimieren.

Um die Anwendbarkeit und die Nützlichkeit der Richtlinien noch weiter zu validieren sollten die Richtlinien in weiteren Fallstudien angewendet werden. Insbesondere sollten die zur Ableitung verwendeten Use Cases dann mehr und feinere Regeln enthalten. Die Vorteile der Richtlinien gegenüber einem ad-hoc Ansatz zur Erstellung von High Level Statecharts sollten in einem Experiment evaluiert werden. Eine offene Fragestellung, bei der noch zusätzlicher Forschungsbedarf besteht ist, ob die Anwendung von Sprachmustern in der Use Case Beschreibung, die Ableitung der High Level Statecharts noch weiter erleichtern kann. Dazu werden momentan Richtlinien entworfen, die den Einsatz solcher Sprachmuster bei der Use Case Erstellung vorsehen.

5 Anhang

Im Folgenden sind das Use Case Diagram der Fenstersteuerung und die textuelle Beschreibung der Use Cases dieses Funktionsblock beschrieben. Diese bilden die Grundlage für die in dem Bericht beschriebenen Beispiele.

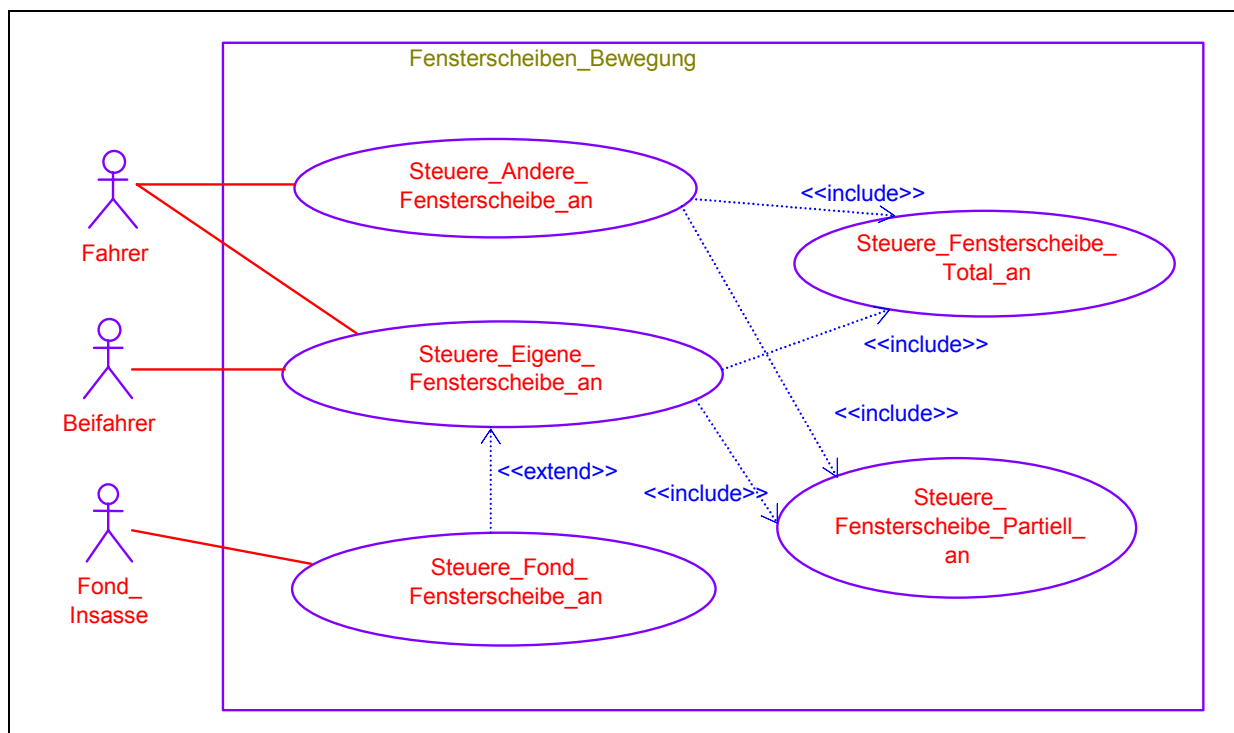


Abbildung 14: Use Case Diagramm des Funktionsblocks Fenstersteuerung

Name	Steuere andere Fensterscheibe an
Aktor	FahrerIn
Ziel	Aktor stellt die Position einer seitlichen Fensterscheibe im Fahrzeug ein, die nicht seine eigene ist
Vorbedingungen	Keine
Beschreibung	1a) Aktor stellt seitliche Fensterscheibe partiell ein (Use Case „ <i>Steuere Fensterscheibe partiell an</i> “) Oder 1b) Aktor stellt seitliche Fensterscheibe total ein (Use Case „ <i>Steuere Fensterscheibe total an</i> “) [Ausnahme für 1a) und 1b): beliebiger Insasseln stellt seine eigene seitliche Fensterscheibe ein]

Ausnahmefälle	<ul style="list-style-type: none"> • Siehe inkludierte Use Cases (siehe Abbildung 14) • beliebiger Insasseln stellt seine eigene seitliche Fensterscheibe ein: bei FahrerIn und BeifahrerIn: Use Case „Steuere eigene Fensterscheibe an“, bei Insasseln hinten: Use Case „Steuere Fond Fensterscheibe an“
Regeln	Siehe inkludierte Use Cases (siehe Abbildung 14)
Qualitätsanforderungen	Siehe inkludierte Use Cases (siehe Abbildung 14)
Eingänge	Siehe inkludierte Use Cases (siehe Abbildung 14)
Ausgänge	Siehe inkludierte Use Cases (siehe Abbildung 14)
Nachbedingungen	Fensterscheibenposition verändert

Name	Steuere eigene Fensterscheibe an
Aktor	FahrerIn oder BeifahrerIn
Ziel	Aktor stellt die Position der eigenen seitlichen Fensterscheibe ein
Vorbedingungen	Keine
Beschreibung	1a) Aktor stellt seitliche Fensterscheibe partiell ein (Use Case „ <i>Steuere Fensterscheibe partiell an</i> “) Oder 1b) Aktor stellt seitliche Fensterscheibe total ein (Use Case „ <i>Steuere Fensterscheibe total an</i> “) [Ausnahme für 1a) und 1b): FahrerIn stellt die Position der betroffenen seitlichen Fensterscheibe ein]
Ausnahmefälle	Siehe inkludierte Use Cases (siehe Abbildung 14) FahrerIn stellt die Position der betroffenen seitlichen Fensterscheibe ein: Use Case „ <i>Steuere andere Fensterscheibe an</i> “
Regeln	Siehe inkludierte Use Cases (siehe Abbildung 14)
Qualitätsanforderungen	Siehe inkludierte Use Cases (siehe Abbildung 14)
Eingänge	Siehe inkludierte Use Cases (siehe Abbildung 14)
Ausgänge	Siehe inkludierte Use Cases (siehe Abbildung 14)
Nachbedingungen	Fensterscheibenposition verändert

Name	Steuere Fond Fensterscheibe an
Aktor	Insasseln hinten
Ziel	Aktor stellt die Position der eigenen seitlichen Fensterscheibe ein
Vorbedingungen	Keine
Beschreibung	Siehe Use Case " <i>Steuere eigene Fensterscheibe an</i> " [Ausnahme: Kindersicherung ist aktiviert]
Ausnahmefälle	Siehe Use Case " <i>Steuere eigene Fensterscheibe an</i> " Kindersicherung ist aktiviert: keine Reaktion
Regeln	Siehe Use Case " <i>Steuere eigene Fensterscheibe an</i> "

Qualitätsanforderungen	Siehe Use Case " <i>Steuere eigene Fensterscheibe an</i> "
Eingänge	Siehe Use Case " <i>Steuere eigene Fensterscheibe an</i> " Kindersicherung
Ausgänge	Siehe Use Case " <i>Steuere eigene Fensterscheibe an</i> "
Nachbedingungen	Fensterscheibenposition verändert

Name	Steuere Fensterscheibenposition partiell an
Aktor	FahrerIn oder BeifahrerIn oder Insasseln hinten
Ziel	Aktor stellt die gewünschte Position der seitlichen Fensterscheibe partiell ein
Vorbedingungen	Keine
Beschreibung	1) Aktor gibt partielle Ansteuerung und Richtung für die Fensterscheibe vor 2) System bewegt Fensterscheibe in die angegebene Richtung [Ausnahme: Fensterendposition erreicht] [Ausnahme: Aktor steuert Fensterscheibe total an] [Ausnahme: Einklemmschutz wird aktiv] [Ausnahme: technisches Problem]
Ausnahmefälle	Fensterendposition erreicht: System reagiert nicht Aktor steuert Fensterscheibe total an: Use Case " <i>Steuere Fensterscheibe total an</i> " Einklemmschutz wird aktiv: sofortiges Herunterfahren der Fensterscheibe in Endposition, keine Unterbrechung möglich technisches Problem: System reagiert nicht
Regeln	Das System muss den Einklemmschutz aktivieren, wenn bei Aufwärtsbewegung keine Fensterscheibenbewegung festgestellt wird und die gewünschte Position noch nicht erreicht ist
Qualitätsanforderungen	Die Bewegung der Fensterscheibe muss innerhalb von 3 Sekunden beendet sein
Eingänge	Fensterscheibenpositionsangabe für <ul style="list-style-type: none"> ○ Bewegungsrichtung hoch/runter ○ Bewegungsart partiell/total ● Aktuelle Fensterscheibenbewegung
Ausgänge	Neue Fensterscheibenposition
Nachbedingungen	Fensterscheibenposition verändert

Name	Steuere Fensterscheibenposition total an
Aktor	FahrerIn oder BeifahrerIn oder Insasseln hinten
Ziel	Aktor stellt die gewünschte Endposition der seitlichen Fensterscheibe ein
Vorbedingungen	Keine
Beschreibung	1) Aktor gibt totale Ansteuerung und Richtung für Fensterscheibe vor

	2) System bewegt Fensterscheibe in die angegebene Endposition [Ausnahme: Akteur steuert Fensterscheibe partiell an] [Ausnahme: Einklemmschutz wird aktiv] [Ausnahme: technisches Problem]
Ausnahmefälle	Akteur steuert Fensterscheibe partiell an: Use Case „ <i>Steuere Fensterscheibe partiell an</i> “ Einklemmschutz wird aktiv: sofortiges Herunterfahren der Fensterscheibe in Endposition, keine Unterbrechung möglich technisches Problem: System reagiert nicht
Regeln	Das System muss den Einklemmschutz aktivieren, wenn bei Aufwärtsbewegung keine Fensterscheibenbewegung festgestellt wird und die gewünschte Position noch nicht erreicht ist
Qualitätsanforderungen	Die Bewegung der Fensterscheibe muss innerhalb von 3 Sekunden beendet sein
Eingänge	Fensterscheibenpositionsangabe für <ul style="list-style-type: none"> ○ Bewegungsrichtung hoch/runter ○ Bewegungsart partiell/total
Ausgänge	Neue Fensterscheibenposition
Nachbedingungen	Fensterscheibe ist in Endposition

6 Literatur

- [HP01] Houdek, Frank and Paech, Barbara: Das Türsteuergerät - eine Beispielspezifikation; IESE-Report Nr. 002.02/D; 2002.
- [DP03] Denger Christian, Paech Barbara, Guidelines to create High Quality Use Cases; IESE-Report to be published; 2003

Dokumenten Information

Titel: Von Use Cases zu Statecharts in
7 Schritten

Datum: 18. Dezember, 2002
Report: IESE-086.02/D
Status: Final
Klassifikation: Public

Copyright 2002, Fraunhofer IESE.
Alle Rechte vorbehalten. Diese Veröffentlichung darf für kommerzielle Zwecke ohne vorherige schriftliche Erlaubnis des Herausgebers in keiner Weise, auch nicht auszugsweise, insbesondere elektronisch oder mechanisch, als Fotokopie oder als Aufnahme oder sonstwie vervielfältigt, gespeichert oder übertragen werden. Eine schriftliche Genehmigung ist nicht erforderlich für die Vervielfältigung oder Verteilung der Veröffentlichung von bzw. an Personen zu privaten Zwecken.