

Erweiterung und formale Verifikation von dynamischen objektorientierten Modellierungsansätzen auf Basis höherer Petri-Netze

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

vorgelegt

der Fakultät für Informatik und Automatisierung

der Technische Universität Ilmenau

vorgelegt am 09.03.2004

verteidigt am 17.09.2004

von

Dipl.-Ing. Olga Fengler

geb. am 12.01.1968 in Moskau

Gutachter: Prof. Dr.-Ing. habil. Ilka Philippow, TU Ilmenau (wiss. Betreuer)
Prof. Dr. rer. nat. Kurt Lautenbach, Universität Koblenz-Landau
Prof. Dr.-Ing. habil. Ulrich Engmann, TU Ilmenau

Vorwort des Verfassers

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin im Institut für Theoretische und Technische Informatik an der Technischen Universität Ilmenau.

Grundlage dieser Arbeit sind unter anderem eine Reihe Diplomarbeiten, die von mir betreut wurden. Viele meiner Gedanken zur vorliegenden Dissertation wurden in diese Arbeiten eingebracht und unter meiner Anleitung entwickelt. Es handelt sich um folgende Diplomarbeiten:

- „Entwicklung und Untersuchung von gefärbten Zustandsdiagrammen“ von Alexei Slavianov [Slav 2001],
- „Entwicklung und Untersuchung von gefärbten Zeitintervall-Aktivitätsdiagrammen“ von Yuri Riabov [Riab 2001],
- „Transformation von Zustands-, Aktivitäts- und Sequenzdiagrammen in einander über eingeschränkte höhere Petri-Netze“ von Christian Rang [Rang 2001],
- „Entwicklung und Untersuchung von Gefärbten Sequenzdiagrammen“ von Christian Amberg [Ambe 2001],
- „Toolmäßige Implementierung von Modellierungsmethoden“ von Sergey Barun [Baru 2002],
- „Verifikation von gefärbten Zeitintervall-Zustands-, Aktivitäts- und Sequenzdiagrammen“ von Alexei Glazounov [Glaz 2002],
- „Erweiterung und Implementation von Verifikationsmethoden für gefärbte Zeitintervall-Petri-Netze“ von Jörg Roth [Roth 2002] und
- „Modellierungsmethodik mit gefärbten Zeitintervallzustands-, Aktivitäts- und Sequenzdiagrammen“ von Andrei Berg [Berg 2002].

Hiermit möchte ich mich für die geleistete Arbeit und die kooperative Zusammenarbeit bei allen bedanken.

Die wesentlichen Ergebnisse zur Entwicklung der Zielstellung der vorliegenden Dissertation und der ihr zugrunde liegenden Diplomarbeiten entstanden bereits zu Beginn der Bearbeitung der Thematik. Sie waren auch Gegenstand mehrerer eigener Veröffentlichungen und gingen in die Zwischen- und Abschlussberichte eines DFG-Themas ein. Deshalb enthält der Stand der Technik weniger Literaturverweise aus dem Zeitraum unmittelbar vor der Fertigstellung der Arbeit, wobei aber die wesentlichen Entwicklungen im Umfeld weiter verfolgt wurden.

Ich möchte mich hier bei allen Mitarbeitern der Fachgebiete Rechnerarchitektur und Prozessinformatik des oben genannten Institutes bedanken, die mich während dieser Zeit fachlich durch Beratung und als Gesprächspartner unterstützt haben.

Mein besonderer Dank gilt meiner Betreuerin Univ.-Prof. Dr.-Ing. habil. Ilka Philippow, die mich in dieser Zeit bei der Erstellung meiner Arbeit durch wertvolle fachliche Betreuung unterstützt hat.

Besonders bedanke ich mich weiterhin bei Herrn Dr.-Ing. Bernd Däne für die Beratungen zu allen meinen Fragen und als Konsultationspartner bei der Fertigstellung der schriftlichen Arbeit.

Ilmenau, im März 2004

Olga Fengler

Inhaltsverzeichnis

	Seite
1 Einführung und Motivation	11
2 Modellierung von komplexen verteilten eingebetteten Echtzeitsystemen	13
2.1 Komplexe verteilte eingebettete Echtzeitsysteme	14
2.2 Zur Entwicklung der dynamischen Modellierung	16
2.2.1 Automatenbasierte Modelle	16
2.2.2 Message Sequence Charts	17
2.2.3 Petri-Netz-basierte Modelle	19
2.2.4 Kontinuierliche Modelle	21
2.2.5 Hybride Modelle	22
2.3 Konzepte der Objekttechnologie und der UML als Standard für die objekt-orientierte Modellierung	24
2.3.1 Beschreibungselemente der UML	26
2.3.2 Diagrammübersicht	27
2.3.2.1 Anwendungsfalldiagramm	27
2.3.2.2 Klassendiagramm	29
2.3.2.3 Zustandsdiagramm	30
2.3.2.4 Aktivitätsdiagramm	33
2.3.2.5 Sequenzdiagramm	35
2.3.2.6 Kollaborationsdiagramm	37
2.3.2.7 Implementierungsdiagramme	38
2.4 Höhere Petri-Netze als Beschreibungsmittel	39
2.4.1 Grundlegende Definitionen	39
2.4.2 Zeitbewertete Petri-Netze	41
2.5 Algorithmen zur Überführung von UML-Diagrammen in Petri-Netze	42
2.6 Verifikationsmethoden	48
2.7 Schlussfolgerungen und Ziele	50
3 Gefärbte und zeitbewertete Diagramme	52
3.1 Beschreibung von Varianten eines Modells in einem Diagramm	53
3.2 Gefärbte Zustandsdiagramme	55
3.2.1 Definition	56
3.2.2 Syntaxerweiterung	56
3.2.2.1 Der gefärbte Zustand	57
3.2.2.2 Die gefärbte Transition	59
3.2.3 Zeitbewertungen	61
3.2.4 Funktionsweise	62

3.1	Gefärbte Aktivitätsdiagramme	63
3.3.1	Definitionen zum Diagramm	63
3.3.2	Syntaxerweiterung	65
3.3.2.1	Die gefärbte Aktion	65
3.3.2.2	Die gefärbte Transition	65
3.3.2.3	Erweiterung von anderen Elementen im Aktivitätsdiagramm	67
3.3.3	Berücksichtigung von Zeitbedingungen	68
3.3.4	Funktionsweise	69
3.2	Gefärbte Sequenzdiagramme	72
3.4.1	Definition	73
3.4.2	Syntaxerweiterung	73
3.4.3	Zeitbewertung	76
3.4.4	Funktionsweise	77
4	Modellierung mit gefärbten Diagrammen	80
4.1	Gefärbtes Aktivitätsdiagramm	80
4.2	Gefärbtes Zustandsdiagramm	82
4.3	Gefärbtes Sequenzdiagramm	87
5	Transformationen zwischen gefärbten Diagrammen über höhere Petri-Netze	93
5.1	Elementevergleich von gefärbten Aktivitäts-, Zustands-, und Sequenzdiagrammen	94
5.2	Transformation von gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen in höhere Petri-Netze	101
5.3	Eigenschaften von eingeschränkten, durch Transformation entstandenen Petri-Netzen	108
5.4	Transformation eingeschränkter Petri-Netze in Zustands-, Aktivitäts- und Sequenzdiagramme	109
5.5	Transformation von Zustands- Aktivitäts- und Sequenzdiagrammen über Gefärbte Petri-Netze ineinander am Beispiel von einem Messsystem	115
5.6	Toolrealisierung	127
6	Verifikation von gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen	130
6.1	Gefärbte Zeitintervall-Petri-Netze	131
6.2	Die Verifikationsmethode	133
6.3	Eigenschaften von Gefärbten Zeitintervall-Petri-Netzen	136
6.4	Toolrealisierung	137
7	Zusammenfassung und Ausblick	141
	Literaturverzeichnis:	145

Abbildungverzeichnis

	Seite
Abbildung 1: Struktur eines eingebetteten Echtzeitsystems	14
Abbildung 2: Struktur eines verteilten eingebetteten Rechnersystems	15
Abbildung 3: State Chart für ein Messsystem	17
Abbildung 4: MSC-96	18
Abbildung 5: Darstellungsmöglichkeiten von asynchroner Nebenläufigkeit	20
Abbildung 6: Hybrides System	22
Abbildung 7: Ereignisdiskrete Generierung von Sprüngen einer kontinuierlichen Prozessgröße	23
Abbildung 8: Entwicklung der UML nach [Camb 98] mit aktueller Erweiterung	25
Abbildung 9: Einteilung nach statischen und dynamischen Aspekten	26
Abbildung 10: Use-Case-Diagramm für ein Lasermesssystem	28
Abbildung 11: Messkopf	28
Abbildung 12: Klassendiagramm	29
Abbildung 13: Zustandsdarstellung	31
Abbildung 14: Verfeinerter UND-Zustand	31
Abbildung 15: Verfeinerter ODER-Zustand	31
Abbildung 16: Synchronisations-Zustand	31
Abbildung 17: Einfacher Zustandsübergang, UND- bzw. ODER-Synchronisationen, parallele Verzweigung und Entscheidungsvarianten	32
Abbildung 18: Zustandsdiagramm für ein Messsystem	33
Abbildung 19: Elemente des Aktivitätsdiagrammes: Zustände und Zustandsübergänge, Parallelitäts- und Entscheidungskonzepte	34
Abbildung 20: Aktivitätsdiagramm	35
Abbildung 21: Objekterzeugung, Objektzerstörung, Selbstinteraktion	36
Abbildung 22: Verschiedene Nachrichtentypen in Sequenzdiagrammen	36
Abbildung 23: Parallelität, Entscheidung, parallele Iteration	37
Abbildung 24: Sequenzdiagramm des Beispiels Messsystem	37
Abbildung 25: Kollaborationsdiagramm für das Messsystem	38
Abbildung 26: Komponenten- und Verteilungsdiagramm	39
Abbildung 27: Gefärbtes Zeitintervall-Petri-Netz	42
Abbildung 28: Beispiel zur Faltung von Zuständen und Transitionen	53
Abbildung 29: Beispiel zu Faltung von Aktivitäten und Transitionen	54
Abbildung 30: Beispiel zu Faltungsmöglichkeiten in Sequenzdiagrammen	54
Abbildung 31: Notationsvarianten für gefärbte Belegungen von Aktionen, Zuständen und Instanzen	55
Abbildung 32: Weitere Notationsvariante für die Darstellung von gefärbten Aktionen, Zuständen und Instanzen	55
Abbildung 33: Beispiel für ein gefärbtes Zustandsdiagramm	56
Abbildung 34: Beispiel zu einem gefärbten Zustand	57

Abbildung 35:	Ein gefärbtes Zustandsdiagramm aus dem Beispiel „Messsystem“	57
Abbildung 36:	Beispiel zu einem gefärbten Startzustand	58
Abbildung 37:	Beispiel für einen gefärbten History-Zustand	58
Abbildung 38:	Beispiel zur ODER-Verfeinerung in gefärbten Zustandsdiagrammen	59
Abbildung 39:	UND-Verfeinerung für Zustand1	59
Abbildung 40:	Beispiel für eine erweiterte Transitionsspezifikation	60
Abbildung 41:	Beispiel für eine komplexe Transition, welche eine Verzweigung darstellt	60
Abbildung 42:	Darstellungsbeispiel von Synchronisationselementen	60
Abbildung 43:	Verschicken desselben Ereignisses an verschiedene Objekte	61
Abbildung 44:	Zeitereignisse im gefärbten Zustandsdiagramm	61
Abbildung 45:	Beispiel für ein gefärbtes Zustandsdiagramm	62
Abbildung 46:	Ein weiteres Beispiel für ein gefärbtes Zustandsdiagramm	63
Abbildung 47:	Ausschnitt aus einem komplexeren gefärbten Aktivitätsdiagramm	64
Abbildung 48:	Angabemöglichkeiten von Kapazitäten in gefärbten Aktionen	65
Abbildung 49:	Notationsbeispiele für eine Transitionsfunktion	66
Abbildung 50:	Beispiel für einen OR-Split	66
Abbildung 51:	Beispiel für einen OR-Join	67
Abbildung 52:	Beispiel für einen AND-Split	67
Abbildung 53:	Beispiel für einen AND-Join	67
Abbildung 54:	Beispiel für einen Start- und einen Endzustand	68
Abbildung 55:	Verfeinerter Aktivitätszustand	68
Abbildung 56:	Notation von Zeitereignissen (Intervalle oder Werte)	69
Abbildung 57:	Berücksichtigung der Zeit beim Modellieren	69
Abbildung 58:	Funktionsweise für ein Beispiel eines gefärbten Aktivitätsdiagrammes	70
Abbildung 59:	Ausführung von Aktion A für die rote Farbe	70
Abbildung 60:	Ausführung von Aktion A für die grüne Farbe	71
Abbildung 61:	Beispiel für ein falsch entworfenes Diagramm (Regel 1)	71
Abbildung 62:	Beispiel für ein falsch entworfenes Diagramm (Regel 2)	72
Abbildung 63:	Beispiel für ein falsch entworfenes Diagramm (Regel 3, zu viele Marken)	72
Abbildung 64:	Beispiel für ein gefärbtes Sequenzdiagramm	73
Abbildung 65:	Repräsentation von gefärbten Objekten	74
Abbildung 66:	Farbabhängige Nachrichtentypen	75
Abbildung 67:	Beispielhafte Zeitangaben für gefärbte Sequenzdiagramme	77
Abbildung 68:	Beispielhafte Verläufe von Lebenslinien für zwei verschiedene Farben	77
Abbildung 69:	Beispiel zur Funktionsweise	79
Abbildung 70:	Beispiel zur Lebenslinienaufspaltung	79
Abbildung 71:	Aktivitätsdiagramm entsprechend UML zur Modellierung des Messsystems	81
Abbildung 72:	Gefärbtes Aktivitätsdiagramm zur Modellierung des Messsystems	83
Abbildung 73:	Zustandsdiagramm nach UML zur Modellierung des Messsystems	85
Abbildung 74:	Gefärbtes Zustandsdiagramm zur Modellierung des Messsystems	86

Abbildung 75:	Sequenzdiagramm nach UML zur Modellierung des Messsystems	88
Abbildung 76:	Gefärbtes Sequenzdiagramm zur Modellierung des Messsystems	89
Abbildung 77:	Sequenzdiagramm nach UML zur Modellierung der Bewegungen des Messtischs	90
Abbildung 78:	Gefärbtes Sequenzdiagramm zur Modellierung der Bewegungen des Messtischs	91
Abbildung 79:	Lebenslinienaufspaltung im gefärbten Sequenzdiagramm	101
Abbildung 80:	Platzaufspaltung und Platzvervielfältigung eines Startplatzes, eines Zwischenplatzes und eines Endplatzes	109
Abbildung 81:	Transformationsschema für einen Startplatz, Endplatz, Zustandswechsel und Pseudozustandswechsel aus dem Gefärbten Petri-Netz	110
Abbildung 82:	Transformationsschema für Nachrichten mit Konvertierungsfunktionen, die Farbkonstanten und vordefinierte Farbvariablen ANY enthalten	111
Abbildung 83:	Transformationsschema für die Bausteine des Gefärbten Petri-Netzes, die der Parallelität entsprechen	111
Abbildung 84:	Transformationsschema für die Bausteine des Gefärbten Petri-Netzes, die die Entscheidungskonstruktion darstellen	112
Abbildung 85:	Modellierung des Messsystems mittels Zustandsdiagramm	116
Abbildung 86:	Verfeinerung des Zustandes „Achse Bewegung“	116
Abbildung 87:	Nach der Transformation entstandenes Gefärbtes Petri-Netz	117
Abbildung 88:	Transformation von Bausteinen des Petri-Netzes in das Aktivitätsdiagramm	118
Abbildung 89:	Transformation von Bausteinen des Petri-Netzes in das Sequenzdiagramm	118
Abbildung 90:	Gefärbtes Aktivitätsdiagramm des Messsystems	119
Abbildung 91:	Gefärbtes Sequenzdiagramm des Messsystems	120
Abbildung 92:	Aufspaltung eines Sequenzdiagramms in mehrere Bausteine	121
Abbildung 93:	Ersetzung der Aktivitätsdiagrammbausteine durch Bausteine des Petri-Netzes	122
Abbildung 94:	Aus dem Aktivitätsdiagramm entstandenes Gefärbtes Petri-Netz	123
Abbildung 95:	Transformation der Gefärbten Petri-Netz-Bausteine in Bausteine des gefärbten Sequenzdiagramms	124
Abbildung 96:	Aus dem Aktivitätsdiagramm entstandenes gefärbtes Sequenzdiagramm	125
Abbildung 97:	Transformation der aus dem Aktivitätsdiagramm entstandenen Petri-Netz-Bausteine in das gefärbte Zustandsdiagramm	126
Abbildung 98:	Aus dem Aktivitätsdiagramm entstandenes gefärbtes Zustandsdiagramm	127
Abbildung 99:	Erstellen der Petal-Dateien	128
Abbildung 100:	Einlesen einer Petal-Datei in einen Konverter	128
Abbildung 101:	Überprüfung eines Modells (Messsystem) in Visual Object Net++	129
Abbildung 102:	Beispiel für ein Gefärbtes Zeitintervall-Petri-Netz	131
Abbildung 103:	Tabellarische Darstellung der Konvertierungsfunktion mit Zuordnung von Zeitintervallen	131

Abbildung 104:	Gefärbtes Petri-Netz	134
Abbildung 105:	Zeiten für die Transitionsequenz	135
Abbildung 106:	Darstellungsbereich vom Petri-Netz	137
Abbildung 107:	Eigenschaftsfenster	138
Abbildung 108:	Überprüfung der Erreichbarkeit eines Zustandes	138
Abbildung 109:	Eingabe einer Transitionssequenz	139
Abbildung 110:	Ergebnisse der Berechnungen zur Transitionssequenz	140

Tabellenverzeichnis

Tabelle 1:	Vergleich von MSC-96,-2000 und UML-SD	19
Tabelle 2:	Aktivitätsdiagramm- und Petri-Netz-Bausteine	43
Tabelle 4:	Sequenzdiagramm- und Petri-Netz-Bausteine	44
Tabelle 3:	Zustandsdiagramm- und Petri-Netz-Bausteine	45
Tabelle 5:	Elementvergleich	95
Tabelle 6:	Transformation von Elementen des gefärbten Aktivitätsdiagramms in Elemente des Gefärbten Petri-Netzes	102
Tabelle 7:	Transformation von Elementen gefärbter Zustandsdiagramme in Elemente Gefärbter Petri-Netze	104
Tabelle 8:	Transformation von Elementen gefärbter Sequenzdiagramme in Elemente Gefärbter Petri-Netze	106
Tabelle 9:	Transformation Gefärbter Petri-Netze in Bausteine gefärbter Zustands- und Sequenzdiagramme	113
Tabelle 10:	Transformation Gefärbter Petri-Netze in Bausteine gefärbter Aktivitäts- und Sequenzdiagramme	114
Tabelle 11:	Zusätzliche Informationen zu dem Zustandsdiagramm	117

Abkürzungverzeichnis

Abkürzungen

OMG	Object Management Group
MSC	Message Sequence Charts
OMT	Object Modeling Technique
SC	State Chart
SD	Sequenzdiagramm
ZD	Zustandsdiagramm
AD	Aktivitätsdiagramm
UML	Unified Modeling Language
PN	Petri-Netz
IPN	Intervall-Petri-Netz
GPN	Gefärbtes Petri-Netz

CPN	Colored Petri Net
SDL	Specification and Design Language
GSD	gefärbtes Sequenzdiagramm
GZD	gefärbtes Zustandsdiagramm
GAD	gefärbtes Aktivitätsdiagramm

Formelzeichen

∨	ODER
&	UND

Kapitel 1

Einführung und Motivation

Der Entwurf von komplexen verteilten eingebetteten Rechnersystemen ist durch die Größe und die Vielzahl von parallel unter Echtzeitbedingungen arbeitenden Komponenten häufig eine große Herausforderung. Als wichtige Eigenschaften von solchen Systemen sind die Nebenläufigkeit, Kommunikation zwischen verschiedenen Teilen, verteilte Steuerung und das Vorhandensein von zeitlichen Restriktionen zu nennen. Mit der schnellen Steigerung der Komplexität von verteilten eingebetteten Rechnersystemen gewinnt die Modellierung immer mehr an Bedeutung. Die Modellierungsmittel müssen einesteils immer leistungsfähiger werden, um diese zunehmende Komplexität zu beherrschen. Andererseits sollten sie ingenieurtechnisch gut handhabbar und praktikabel sein.

Die dynamischen Diagramme, wie Message Sequence Charts (Sequenzdiagramme), State Charts (Zustandsdiagramme) und Activity Diagrams (Aktivitätsdiagramme) stellen weit verbreitete Modellierungsmittel für digitale Systeme dar. Diese Diagramme werden seit einigen Jahren unterstützend im Entwicklungsprozess, besonders in der Spezifikationsphase, eingesetzt. Diese grafischen Beschreibungstechniken bieten die Möglichkeit, verschiedene Aspekte eines Systems darzustellen. Schon im frühen Entwurfsstadium geben sie einen Überblick über die geplante Lösung der Aufgabenstellung. Durch ihre grafische Form und verschiedene Abstraktionsebenen können sie zur Verdeutlichung der Funktion dienen. Diese einfache grafische Repräsentation macht das zu entwerfende Produkt auch für fachfremde Anwender überschaubar. Damit kann frühzeitig die Konsistenz zwischen Anwenderwünschen und Funktionalität

überprüft werden. Diese Diagramme können aber häufig nicht die Komplexität der zu modellierenden Systeme bewältigen und dabei die Übersichtlichkeit der Entwurfsmodelle beibehalten. Es existiert auch keine Überführungsmöglichkeit von einem Beschreibungstyp in einen anderen. Das grafische Konzept wird auch wenig durch einen mathematischen Formalismus unterstützt. Somit sind die Möglichkeiten zur formalen Analyse der Systemeigenschaften nicht oder nur eingeschränkt gegeben.

In dieser Arbeit werden gefärbte dynamische Sequenzdiagramme, Zustandsdiagramme und Aktivitätsdiagramme für die Modellierung von komplexen verteilten Echtzeitsystemen entwickelt. Diese sind auch in der Unified Modeling Language (UML) als ungefärbte Variante enthalten. Es werden dabei nachstehende Ziele verfolgt:

- Reduktion der Komplexität der Darstellung von verteilten Informationsverarbeitungsfunktionen,
- Übernahme von Teilen des etablierten Konzeptes der höheren Petri-Netze auf weit verbreitete Diagrammtypen, die auch in die objektorientierte Softwareentwicklung (Unified Modeling Language) integriert wurden,
- Kompakte Darstellung ähnlicher paralleler Prozesse (Objekte),
- Transformation in Gefärbte Petri-Netze zum Zweck der späteren Verifikation.
- Entwicklung ergänzender Verifikationsmethoden

Mit der Weiterentwicklung zu gefärbten Diagrammen, wie sie in der vorliegenden Arbeit vorgenommen wird, kann das Verhalten von mehreren Objekten (Prozessen) in einem Diagramm gleichzeitig modelliert werden. Die gefärbten Diagrammtypen verfügen über eine Menge von ausreichenden Mitteln für die übersichtliche und eindeutige Darstellung der Komposition von mehreren einfachen Diagrammen und einige zusätzliche Mechanismen für die Abbildung der Abhängigkeiten und Beziehungen zwischen einzelnen Objekten. Die Transformation von diesen Diagrammtypen ineinander erlaubt es, ähnliche Beschreibungsmittel parallel nutzbar zu machen.

In dieser Arbeit wird gezeigt, wie sich die neue Eigenschaft „Farbe“ in die bestehenden Regeln einfügt, wie sich Verifikationsmethoden für Gefärbte eingeschränkte Petri-Netze, die bei der Transformation aus diesen Diagrammtypen entstanden sind, erweitern lassen. Weiterhin werden die entwickelten Algorithmen für die Transformationen der gefärbten Diagramme ineinander beschrieben.

Die Farben, die im wesentlichen verschiedene ähnliche Teilkomponenten modellieren, werden bei der Umwandlung in Gefärbte Petri-Netze, die als Zwischennotation genutzt werden, und weiterhin bei der Transformation in andere Diagrammtypen beibehalten und dienen der übersichtlichen Zuordnung und Identifizierung der einzelnen Komponenten eines Modells während des gesamten Entwurfsprozesses.

Kapitel 2

Modellierung von komplexen verteilten eingebetteten Echtzeitsystemen

Ein Überblick über verteilte eingebettete Echtzeitsysteme stellt die Problembereiche bei der Modellierung von solchen Systemen dar. Nach einer Skizzierung des Aufbaus allgemeiner verteilter und eingebetteter Echtzeitsysteme folgt die Vorstellung wichtiger Besonderheiten, welche bei dem Modellierungsprozess von solchen Systemen zu berücksichtigen sind.

Eine Vorstellung von dynamischen Modellierungsansätzen, unter welchen verschiedene Modelltypen beschrieben werden, stellt einleitend die wichtigsten Modellierungsmittel, welche bei dem Entwurf von verteilten eingebetteten Echtzeitsystemen oft genutzt werden, vor.

Die Darstellung von Konzepten der Objekttechnologie und der UML als Standard für die objektorientierte Modellierung, die eine Vielzahl von Möglichkeiten zum Entwurf von verschiedenen Systemen, darunter auch für verteilte eingebettete Echtzeitsysteme, bietet, bildet einen Schwerpunkt des zweiten Kapitels. Die grundlegenden Beschreibungselemente der UML und einige Vorgehensmodelle werden erläutert. Es wird dabei besonders auf die bei Ingenieuren und Informatikern auch ohne UML weit verbreiteten Diagrammtypen Zustandsdiagramme, Aktivitätsdiagramme und Sequenzdiagramme eingegangen, die in der UML zur Beschreibung des dynamischen Verhaltens dienen.

Da die höheren Zeitintervall-Petri-Netze sich gut zur Formalisierung des Verhaltens verteilter eingebetteter Echtzeitsysteme mit dem Zweck der Verifikation von kausalen und Zeiteigenschaften eignen, werden hier grundlegende Definitionen dieser angegeben und deren Eigenschaften beschrieben. Danach folgt die Behandlung von bekannten Verifikationsmethoden als

Grundlage zur späteren Erweiterung und Integration in den gesamten Entwurfsprozess. Abschließend wird die Notwendigkeit und Motivation zur Erweiterung von dynamischen objekt-orientierten Modellierungsansätzen für den Entwurf von verteilten eingebetteten Echtzeitsystemen behandelt und die Zielstellung der Arbeit abgeleitet.

2.1 Komplexe verteilte eingebettete Echtzeitsysteme

Eingebettete Rechnersysteme haben in vielen Anwendungsbereichen eine hohe Bedeutung. Sie lösen dabei klassische Aufgaben der Steuerungs- und Regelungstechnik bis hin zur komplexen Datenverarbeitung. Dazu gehören Aufgaben zur Erweiterung und Überwachung der Funktionalität und zur Erleichterung des Umganges mit dem einbettenden Systems. „Der Einsatzbereich eingebetteter Systeme vergrößert sich rapide: Nach Schätzungen der New York Times kam 1998 der durchschnittliche Amerikaner mit ca. 60-100 dieser eingebetteten Systeme täglich in Berührung. In modernen Kraftfahrzeugen kommen heute bis zu 60 eingebettete Systeme vor (ABS, Airbagsteuerung, elektronische Fahrwerksteuerung)“ [Schm 99]. Eingebettet zu sein bedeutet, dass Schnittstellen zum umgebenden System existieren. In der Abbildung 1 [NüFeBö 99] ist eine allgemeine Struktur eines eingebetteten Systems dargestellt.

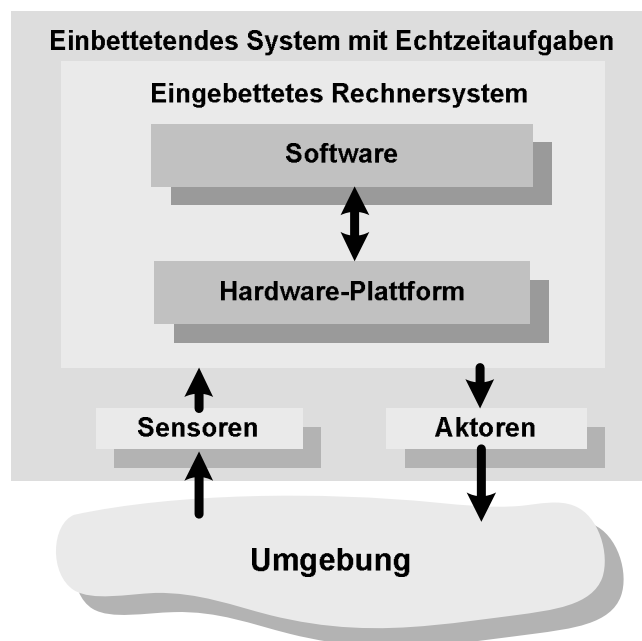


Abbildung 1: Struktur eines eingebetteten Echtzeitsystems

Zum eingebetteten Systemen gehören im Allgemeinen verschiedene Einzelkomponenten, die in eine größere Umgebung integriert sind und diese informationstechnisch beeinflussen. Sie bestehen aus informationsverarbeitender Hard- und Software und sind mit der Umgebung durch Aktoren und Sensoren verbunden. Die Umgebung stellt einen technischen Prozess dar, welcher durch das eingebettete System gesteuert oder geregelt wird [Nütz 00]. Eingebettete Systeme können Standard-Mikroprozessoren und Mikrocontroller und auch moderne Feldbussysteme enthalten [ScWeWe 99, Berm 01, Hind 98]. Die meisten eingebetteten Systeme haben typischerweise Echtzeiteigenschaften. Das bedeutet, dass nicht nur die logische Kor-

rektheit, sondern auch zeitliche Restriktionen Einfluss auf die richtige Funktionalität des eingebetteten Systems haben.

Eingebettete Systeme besitzen folgende typische Eigenschaften:

- Der Rechner ist vollständig in Hardware und Software auf die Anwendung optimiert, weil das Vorhandensein eines fest vorgegebenen übergeordneten technischen Aufgabenspektrums eine besondere Eigenschaft des eingebetteten Systems ist. Dabei wird jedes eingebettete System darauf spezialisiert und es werden nicht nur Software, sondern auch Hardware, also das Computersystem und die weiteren elektronischen Komponenten, an die Anwendung angepasst;
- Echtzeitanforderungen;
- Eventuell sind Teile der Algorithmen in Hardware zu realisieren. Das führt zum Hardware-Software-Co-Design [BaChGi 2000]. Das bedeutet parallele Entwicklung von Hardware und Software mit eventueller Repartitionierung, was zu gleichartigen Spezifikationen von Hardware und Software führt. Um ein funktions- und kostenmäßig optimiertes und fehlerfreies Produkt zu entwickeln, wird dabei der Entwurf von Hardware und Software gemeinsam betrachtet;
- Rechnernetzschnittstelle;
- eventuell Mehr-Prozessor/Mehr-Rechner-Lösung;
- die Schnittstellen zum einbettenden System können digital oder analog sein;
- die Bedienung vom eingebetteten System (Nutzerinterface) ist weitgehend problemangepasst;
- die Softwareentwicklung verlangt meist die Benutzung von Zusatzsystemen.

Die meisten eingebetteten Systeme sind verteilt [Tane 95]. Die Verarbeitung und Speicherung von Informationen und Daten in verteilten Systemen findet nicht nur in einem zentralen Rechnerknoten, sondern in mehreren voneinander mehr oder weniger unabhängigen Einheiten statt. Dabei ist die Kommunikation zwischen den verschiedenen Rechnerknoten ein wichtiger Bestandteil so eines Systems.

Ein verteiltes, über Feldbusssystem vernetztes, eingebettetes System ist auf der Abbildung 2 gezeigt:

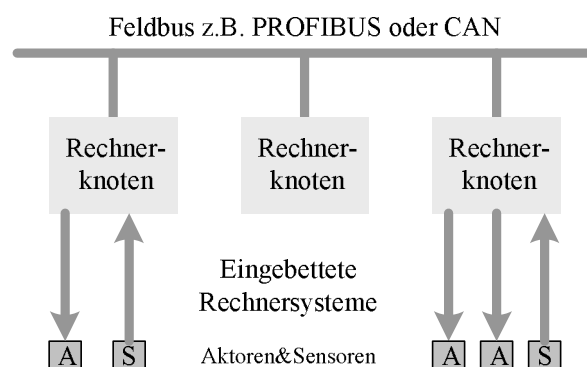


Abbildung 2: Struktur eines verteilten eingebetteten Rechnersystems

Die gezeigte Struktur besteht aus Rechnerknoten, Aktoren (A) und Sensoren (S). Rechnerknoten können aus Bausteinen wie Prozessoren (Mikrocontroller-, Signalprozessor- oder Mikroprozessorschaltkreise), Speichermodulen (RAM, ROM, EEPROM usw.), MSI-(Medium Scale

Integration)-Schaltkreisen (ADC, DAC und/oder MUX und digitale Funktionen) bestehen. Einige Rechnerknoten brauchen keine Verbindung mit Aktoren und Sensoren zu haben, in diesem Fall erfüllen sie nur Verarbeitungs-, Dialog- und Kommunikationsaufgaben. Die verteilten Rechnerknoten können über einen Feldbus, z.B. PROFIBUS oder CAN-Bus (Controller Area Network) kommunizieren.

Bei dem Entwurf von solchen verteilten Systemen [GaVaNaGo 94] ist es wichtig, die Heterogenität der Systemarchitektur und die Komplexität der Aufgabenstellung zu berücksichtigen. Die für den Entwurf bedeutendsten Besonderheiten von verteilten eingebetteten Systemen sind:

- Es besteht Nebenläufigkeit, wenn die Prozesse in einem solchen System sowohl parallel als auch sequentiell in einer beliebigen, teilweise voneinander abhängigen Folge arbeiten.
- Dabei ist auch die Einhaltung von zeitlichen Restriktionen in den parallelen, meist abhängigen Pfaden notwendig.
- Die Abhängigkeiten verlangen Kommunikation und Synchronisation von Prozessen zwischen verschiedenen Teilen des Systems.
- Die Steueralgorithmen sind eventuell verteilt realisiert.

2.2 Zur Entwicklung der dynamischen Modellierung

Um ein System zu entwerfen, braucht man eine Beschreibung der gewünschten Eigenschaften und Funktionalitäten. Dabei ist es wichtig, nur wesentliche und interessierende Systemeigenschaften und Komponenten zu betrachten und von unwichtigen Details zu abstrahieren. Dazu dient der Prozess der Modellbildung, unter welchem die Definition von Basiselementen und deren Kommunikationsregeln, die ein so genanntes Modell darstellen, verstanden wird. Ein Modell soll nicht nur eindeutig und vollständig sein, sondern auch verständlich und offen für weitere Änderungen. Modellierungsansätze unterscheiden sich in Bezug auf die betrachteten Aspekte eines Systems (z.B. Datenmodellierung oder Ablaufmodellierung) und in Bezug auf verschiedene Abstraktionsebenen und lassen sich nach Einsatzgebieten klassifizieren. Im weiteren werden kurz die bekanntesten und für diese Arbeit wichtigsten Modellierungstechniken, welche die Konzepte Nebenläufigkeit und Synchronisation unterstützen, und die verschiedenen Modellklassen dargestellt.

2.2.1 Automatenbasierte Modelle

Für die Modellierung ereignisdiskreter Systeme sind verschiedene automatenbasierte Modelle geeignet. Die zustandsorientierten Modelle, wie z.B. Modelle endlicher Automaten, repräsentieren das System durch so genannte Zustandsgraphen, die als eine Menge von Zuständen und Zustandsübergängen dargestellt werden. Die schrittdiskreten Zustandsänderungen im Zustandsverhalten von Automaten können mit Hilfe verschiedener Graphenarten beschrieben werden: z.B. Automatengraphen oder Zustands-Übergangs-Diagramme.

Zur Spezifikation von eingebetteten Echtzeitsystemen ist es wichtig, ein Modellierungsmittel, welches das Konzept der Hierarchie und die Eigenschaften der Nebenläufigkeit unterstützt, zu

benutzen. Dabei sind die von David Harel [Hare 87] entwickelten State Charts (Zustandsdiagramme) von besonderer Bedeutung. Sie erweitern die endlichen Automaten um hierarchische Verfeinerung, nebenläufige Zustände und Broadcastkommunikation [Hare 98, Teic 97]. In State Charts wurden als weitere Modellierungselemente neben Zuständen und Transitionen solche wie Events, Conditions und strukturierbare Variablen eingeführt. In Abbildung 3 ist das Verhalten eines Messsystems mit Hilfe von Statecharts als Beispiel dargestellt:

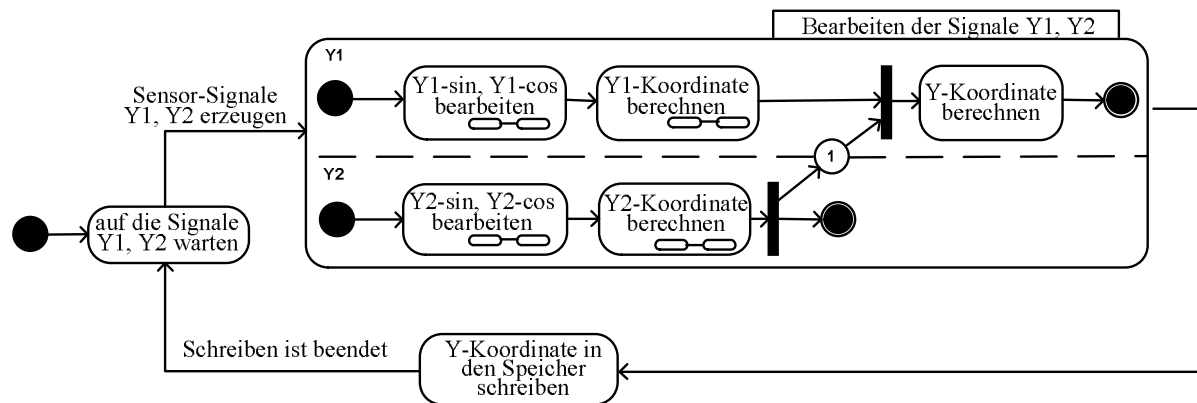


Abbildung 3: State Chart für ein Messsystem

2.2.2 Message Sequence Charts

Die Message Sequence Charts (MSC) [BuMeRo 96, ITU-TS 96, ITU-TS 98, ITU-TS 99] stellen ein weit verbreitetes Modellierungsmittel der Kooperation bzw. des Nachrichtenaustausches in digitalen Systemen dar. Message Sequence Charts (MSC) werden seit einigen Jahren unterstützend im Entwicklungsprozess eingesetzt. Die Hauptanwendung findet sich bei der Darstellung des Kommunikationsverhaltens von Echt-Zeit-Systemen [MaRe 96]. Schon im frühen Stadium geben sie einen Überblick über die geplante Lösung der Aufgabenstellung. Durch ihre grafische Form und verschiedene Abstraktionsebenen können sie zur Verdeutlichung der Funktion dienen. Diese einfache Repräsentation macht das zu entwerfende Produkt auch für fachfremde Anwender überschaubar. Damit kann frühzeitig die Konsistenz zwischen Anwenderwünschen und Funktionalität überprüft werden. MSC's helfen aber nicht nur bei der Validierung und Dokumentation. Durch exakte formale Spezifikationen können sie auch aktiv zur Entwicklung beitragen, indem sie automatisch in andere Formate oder auch Quellcode-Gerüste umgewandelt werden können. Es gibt eine Menge von verschiedenen MSC, die eher als Dialekte fungieren.

MSC wurden ursprünglich von ITU (International Telecommunication Union), einer internationalen Organisation, welche sich mit Richtlinien und Standards im Telekommunikationsbereich beschäftigt, im Jahr 1992 standardisiert (Z.100) [ITU-TS 99]. Eine formale Definition wurde 1994 als Annex-B zur Z.120 hinzugefügt. Besonders große Akzeptanz haben die MSC

seit Einführung des MSC-96-Standards [ITU-TS 96] erhalten, der als der leistungsfähigste und umfangreichste gilt.

Generell beschreiben die MSC eine Funktionalität oder ein Verhalten, dabei steht die Interaktion in Form von Nachrichten im Vordergrund. Der Nachrichtenaustausch zwischen verschiedenen Instanzen, im objektorientierten Umfeld auch Objekte genannt, erfolgt meist asynchron. Die Instanzen können Softwareprozesse, Hardware oder auch Benutzer darstellen. In einem MSC werden zwei Darstellungsdimensionen unterschieden, die *vertikale* Dimension, welche einer klassischen Zeitachse entspricht, während *horizontal* die betrachteten Objekte spezifiziert werden.

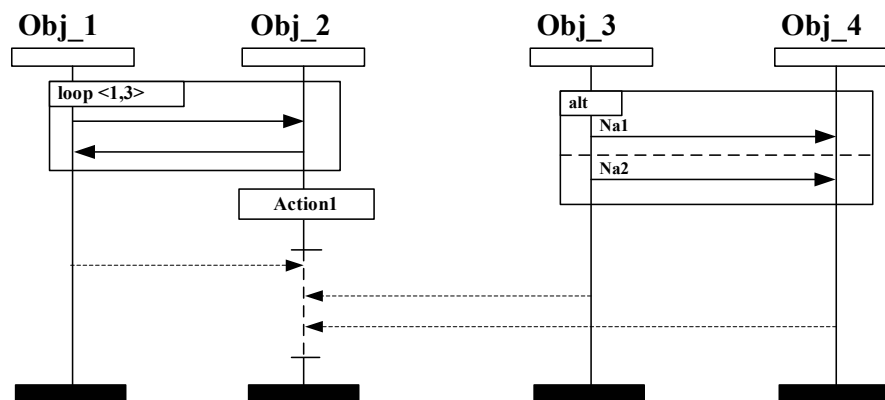


Abbildung 4: MSC-96

In Abbildung 4 ist eine MSC im Standard 96 dargestellt [Ambe 2001]. Sie enthält folgende Elemente:

- *Coregions*, abgegrenzte gestrichelte Lebenslinie, welche ungeordnetes Senden und Empfangen von Nachrichten erlaubt;
- *Actions*, welche meistens zur Dokumentation und zur besseren Verständlichkeit benutzt werden und interne Aktivitäten der Instanz (des Objektes) darstellen;
- *Inline Expressions*, die Alternativen, Parallelität oder Schleifen beschreiben (loop und alt in Abbildung 4).

Die Synchronisationselemente, sogenannte *Conditions* und die Möglichkeiten zur hierarchischen Darstellung (*References* und *Decomposition*) sind in [ITU-TS 96, Krüg 00, Ambe 01, Lang 00] beschrieben.

Im Jahre 2000 wurde eine erweiterte Spezifikation von MSC in Form der MSC-2000 [ITU-TS 99], die eine Obermenge von MSC-96 darstellt und abwärtskompatibel ist, vorgestellt. Die Neuerungen [Krüg 00] beziehen sich auf:

- Verbesserte Strukturkonzepte und Anpassung an die Objektorientierung,
- Datenfelder; Eignung zur automatischen Quellcodegenerierung in verschiedenen Sprachen,
- Hervorhebung des Ablaufs durch Kontrollflusssteuerung, Methodenaufrufe und ähnliches,
- Zeitbindungen; absolute und relative Zeitbestimmungen, Angabe von Zeitintervallen sowie Nebenbedingungen in Abhängigkeit von der Zeit,

- Bedingte Alternativen.

Trotz der höheren Ausdrucksmöglichkeiten, welche die MSC-2000 bieten, existiert leider noch keine vollständige formale Semantik wie in der MSC-96, es gibt noch keine aktuelle Version der Z.120-Annex-B (Formal Semantics of Message Sequence Charts).

Die MSC-96/2000 und die Sequenzdiagramme der Unified Modeling Language (UML), auf welche später eingegangen wird, kann man als Hauptdialekte der Message Sequence Charts ansehen. Die wichtigsten Merkmale und Unterschiede von MSC-96, MSC-2000 und UML-SD sind in der nachfolgenden Tabelle 1 „MSC-Vergleich“ aufgelistet [Krüg 00]:

Tabelle 1: Vergleich von MSC-96,-2000 und UML-SD

Merkmal	MSC-96	MSC-2000	UML-SD
asynchrone Kommunikation	*	*	*
synchrone Kommunikation	-	-	*
Kontrollfluss	-	-	*
Wiederholungen	*	*	(-) ^a
Alternativen (ohne Bedingungen)	*	*	-
bedingte Alternativen	-	*	*
Referenzierung	*	*	-
HMSC's (High-Level MSC's)	*	*	(-) ^b
formale Semantik	*	-	-
Zeitbedingungen	-	(*) ^c	(*) ^c

^a Die Syntax ist nur für die Wiederholung einer einzelnen Nachricht festgelegt. Die Wiederholung eines Abschnittes ist nur informell beschrieben.

^b SD's sind nur ein Beschreibungsmittel in der UML. Beachtet man den gesamten Umfang der UML, kann man Aktivitätsdiagramme anstelle von HMSC's einsetzen.

^c Da die Notationen keine formale Semantik besitzen, haben Zeitangaben nur einen dokumentarischen Nutzen.

Die Sequenzdiagramme der UML unterstützen verschiedene Nachrichtentypen: synchrone, asynchrone sowie Methodenaufrufe und –rückgaben. Dafür besitzen die MSC's-96 bereits die Funktionsvielfalt und eine formale Semantik. Die MSC's-2000 wurden um einige wichtige Merkmale erweitert, die bisher schon in den SD's enthalten waren. So gibt es heutzutage keine großen Unterschiede mehr aus der funktionalen Sicht. Für den praktischen Einsatz zählt hingegen auch die Akzeptanz der Industrie.

2.2.3 Petri-Netz-basierte Modelle

Die in den 60er Jahren von C.A. Petri in seiner Dissertation „Kommunikation mit Automaten“ [Petr 62] entwickelte Theorie von Petri Netzen stellt ein formales Mittel zur Beschreibung von verteilten nebenläufigen Systemen dar. Da es eine Vielzahl verschiedener allgemeiner Veröffentlichungen zu Petri-Netzen gibt, basiert die folgende Darstellung auf einer Zu-

sammenfassung aus [Reis 85, Star 78, Jens 92, Schn 93, Feng 88, RoWi 91, KöQu 88, Kö-Qu89]. Mittlerweile sind Petri-Netze ein anerkannter Beschreibungsformalismus, der sich erfolgreich in verschiedenen Gebieten der Informatik, Steuerungs- und Automatisierungstechnik, der Produktionsmodellierung und beim Workflow-Entwurf zur Modellierung anwenden lässt. Da eine deutliche Tendenz zur Ersetzung bzw. Ergänzung algebraischer Beschreibungsmittel durch grafische existiert, gewinnen solche Beschreibungstechniken wie Petri-Netze bei der Modellierung von parallel ablaufenden Prozessen und Systemen immer mehr an Bedeutung. Mit Hilfe von Petri-Netzen werden kausale Zusammenhänge von diskreten Systemen beschrieben. In Petri-Netzen werden zwei Arten von Netzknoten verwendet: Transitionen und Plätze, dabei wird die Systemstruktur über gerichtete Kanten, die Plätze und Transitionen miteinander verbinden, festgelegt. Die Veränderung der Markierung durch Schalten von Transitionen bildet die Dynamik des modellierten Systems ab. Die Beschreibung nebenläufiger Prozesse wird mit Petri-Netzen besonders günstig möglich. In der Abbildung 5 (nach [Abel 88]) ist asynchrone Nebenläufigkeit, modelliert mittels globalen Zustandsgraphen und mittels Petri-Netz, dargestellt. Im Laufe der Entwicklung wurden Konzepte zur Beschreibung von Nebenläufigkeit auch in andere Beschreibungsmittel, z.B. auch Automaten, integriert.

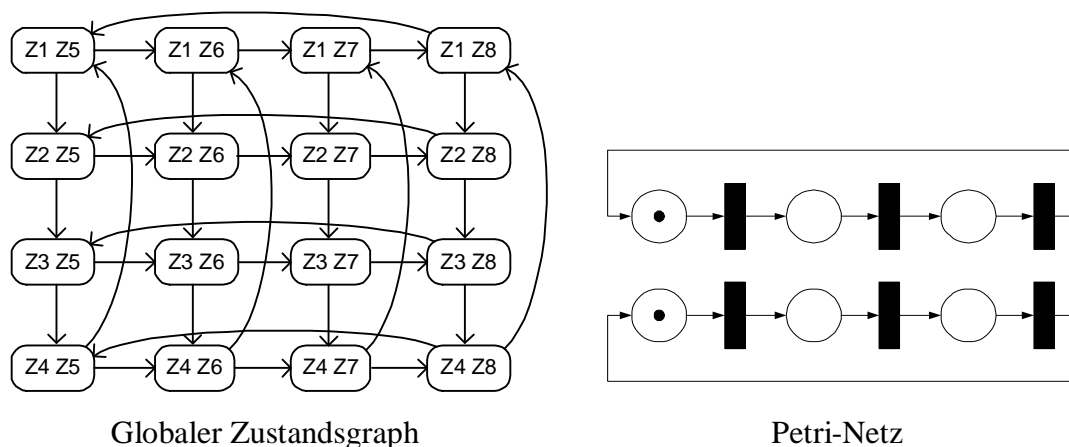


Abbildung 5: Darstellungsmöglichkeiten von asynchroner Nebenläufigkeit

Verschiedene Erweiterungen der einfachen Petri-Netze unterstützen die Beschreibung von komplizierterem Verhalten der modellierten Systeme. Diese Erweiterungen beziehen sich auf die Einführung von höheren Petri-Netzen (z.B. Gefärbten Netzen: Colored Petri Nets, CPN) [Jens 92, Jens 95, JeRo 91], welche in Unterschied zur einfachen Petri-Netzen (z.B. Platz-Transitions-Netzen) nicht anonyme Marken sondern attributierte Marken verwenden. Diese Klasse von Netzen stellt ein adäquates Modellierungsmittel von komplexen Systemen dar. Die Gefärbten Petri-Netze lassen sich in Platz-Transitions-Netze transformieren, was die Anwendung von für diese Netzklasse entwickelten Analysemethoden ermöglicht. Als eine wichtige Erweiterung ist die Einführung von Zeitkonzepten [KöQu 89] zu nennen. Die Zeitwerte oder Zeitintervalle können zu Transitionen, Plätzen, Kanten oder Marken zugeordnet werden und beschreiben damit die Zeiteigenschaften der zu modellierenden Prozesse, Das ermöglicht auch die Untersuchung auf gewünschtes oder unerwünschtes Zeitverhalten. Die Einführung von Hierarchiekonzepten [HuJe 90] erlaubt hierarchische Anordnungen bei der Modellierung von komplexen Systemen.

Die in [DaAl 87] beschriebenen kontinuierlichen Petri-Netze führten in die bis dahin diskreten klassischen Petri-Netze einen kontinuierlichen Modellierungsansatz ein. Dieser Ansatz wurde weiter in zahlreichen Arbeiten [Flau 97, DeSc 97, DeKo 98, Wiet 98, Drath 99, Chou 99] weiterentwickelt. Dabei wurden so genannte hybride Petri-Netze eingeführt, welche sowohl diskretes als auch kontinuierliches Verhalten von verschiedenen Teilen eines Systems zu modellieren gestatten.

Die Petri-Netze sind dank ihrer einfachen grafischen Darstellung und dem gut entwickelten mathematischen Analyseapparat in wissenschaftlichen Arbeiten ziemlich weit verbreitet. Der umfassende Einsatz als Beschreibungsmittel in der Praxis ist weniger zu beobachten. Das liegt vermutlich gerade daran, dass sie sehr formal und mathematisch orientiert verwendet werden können und sollten. Demzufolge sind oft Ansätze erfolgreich, wo Petri-Netz-Konzepte in verbreitete Beschreibungsmittel integriert werden und wo mit Petri-Netzen andere Beschreibungsmittel formalisiert und der Verifikation zugänglich gemacht werden. Diese beiden Konzepte werden in der vorliegenden Arbeit verfolgt.

2.2.4 Kontinuierliche Modelle

Die kontinuierlichen Modelle, in welchen sich die Werte der Zustandsvariablen innerhalb eines endlichen Zeitintervalls unendlich oft ändern und einen beliebigen Wert annehmen können, werden vorwiegend mit Differenzialgleichungen beschrieben. Diese beschreiben die Zusammenhänge zwischen Funktionen einer oder mehrerer Eingangsvariablen und deren Ableitungen. Dabei kommen verschiedene Gleichungen (lineare und nichtlineare Gleichungen, partielle Gleichungen, welche mehrere unabhängige Variablen haben, Differenzialgleichungen mit expliziter und impliziter Schreibweise) in Betracht.

Die häufig für die Modellierung von kontinuierlichen Systemen verwendeten Gleichungen sind explizite Differentialgleichungen, die aus gewöhnlichen linearen oder nichtlinearen Differentialgleichungen 1. Ordnung und aus algebraischen Gleichungen in expliziter Form bestehen:

$$\dot{x} = f(x, u, t)$$

$$y = g(x, u, t)$$

Diese Gleichungen beschreiben ziemlich adäquat viele kontinuierliche Systeme. Die Lösung solcher Gleichungssysteme geschieht mit relativ einfachen numerischen Verfahren.

Manche Systeme lassen sich aber mit der impliziten Form leichter beschreiben, die eine weitere Möglichkeit zum Modellieren von solchen Systemen darstellt. Sie besteht genau wie die oben genannte aus linearen oder nichtlinearen Differentialgleichungen 1. Ordnung und aus algebraischen Gleichungen und ist in impliziter Form angegeben:

$$F(x, \dot{x}, u, t) = 0$$

Dabei benötigt die Lösung von solchen Gleichungssystemen aufwendige numerische Verfahren sowie eine umfangreichere Spezifikation der Anfangsbedingungen. Deshalb existieren nur wenige Simulationswerkzeuge, die solche Modelle unterstützen.

Wenn bei der Modellierung von kontinuierlichen Systemen neben zeitlichen auch räumliche Prozesse auftreten, können solche Systeme mit partiellen Differentialgleichungen beschrieben werden. Wegen der aufwendigeren Lösungsverfahren und speziellen Einsatzgebiete werden diese von den meisten kontinuierlichen Simulationswerkzeugen nicht unterstützt.

2.2.5 Hybride Modelle

Die zustandsorientierten und kontinuierlichen Systeme lassen sich durch ereignisdiskrete oder kontinuierliche Modelle beschreiben. Bei der Modellierung von komplexen eingebetteten Echtzeitsystemen kommt es jedoch häufig vor, dass verschiedene Teile dieser Systeme sowohl kontinuierliches als auch diskretes Verhalten aufweisen und miteinander Wechselwirkungen besitzen. Erstere beziehen sich gewöhnlich auf physikalische Prinzipien, letztere dagegen auf digitale Sensoren und Aktoren, digitale Logik oder Softwarecode. Ein typisches Beispiel sind Echtzeitsysteme, in denen physikalische Prozesse durch Regel- und Steuerungsalgorithmen beeinflusst werden, die in einem eingebetteten System realisiert werden. Zur Beschreibung von solchen Systemen werden so genannte hybride Modelle benutzt, welche dadurch gekennzeichnet sind, dass sie sowohl diskrete als auch kontinuierliche Zustandsvariablen beinhalten.

Die Spezifik von hybriden Systemen besteht dabei darin, dass unterschiedliche Teile in einem solchen System sich wechselseitig beeinflussen (Abbildung 6), was aber schwer zu modellieren ist.

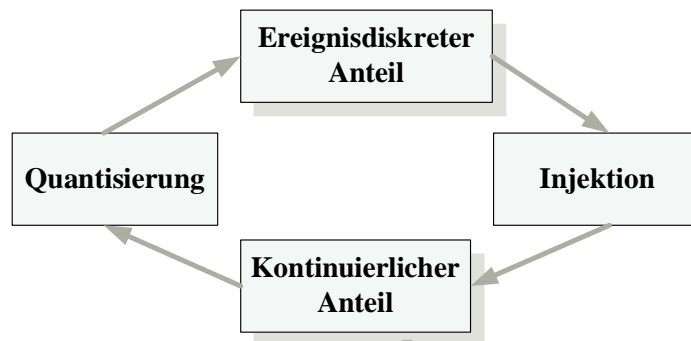


Abbildung 6: Hybrides System

Verschiedene Ansätze, welche sich zur Modellierung von hybriden Systemen eignen, sind meist Erweiterungen der bekannten Automatenmodelle, von Transitionssystemen, Petri-Netzen und temporalen Logiken. Zu anwendungsgerechten Ansätzen zählen:

- Hybride Automaten [AlCoHe 93, Henz 96], welche die Verallgemeinerung von endlichen Automaten, bei denen ereignisdiskrete und kontinuierliche Zustandsvariablen zugelassen sind, darstellen. Es handelt sich dabei genau so wie bei endlichen Automaten um diskrete Zustände, in welchen sich aber auch kontinuierliche Zustandsvariablen befinden können, und um diskrete Zustandsübergänge. Während sich ein hybrider Automat in einem Knoten befindet, können Zustandsänderungen durch diskrete oder

kontinuierliche Änderungen von Zustandsvariablen stattfinden oder sie finden durch den diskreten Wechsel eines Knotens statt. Trotz guter grafischer Darstellungsform und einfacher Semantik ist diese Klasse wegen eingeschränkter Strukturierungsmöglichkeiten nicht ausreichend zur Modellierung von komplexen hybriden Systemen geeignet [HeRu 98].

- Hybride State Charts [KePn 92], die auf den diskreten State Charts basieren und um kontinuierliche Elemente und ein verbessertes Zeitkonzept erweitert sind. Dabei bleibt die Struktur von diskreten State Charts mit Basiszuständen mit den möglichen AND- oder XOR-Verfeinerungen sowie das Transitionskonzept erhalten. Als Neuerung kommt die Einführung der Unterteilung von globalen Zustandsvariablen in diskrete und kontinuierliche Variable dazu. Solange der jeweilige Basiszustand aktiv ist, können kontinuierlichen Variablen durch Angabe von Differenzial-Gleichungen kontinuierlich verändert werden. Die Transitionsbedingungen dagegen dürfen nur diskrete Variablen enthalten. Auf Grund der grafischen Darstellung und hierarchischen Strukturierbarkeit eignen sich die hybriden State Charts gut zur Darstellung von verteilten nebenläufigen Systemen.
- Hybride Petri-Netze, eingeführt von [AlDaBa 91, DaAl 92] und weiterentwickelt in den Arbeiten [Wiet 97, Drat 99]. Sie wurden zur Darstellung von hybriden Systemen entwickelt und erweitern die klassischen Platz-Transition-Netze um ein Kontinuitätskonzept. Die Erweiterung auf kontinuierliche Elemente bezieht sich auf Plätze und Transitionen, dabei bleiben in diesem Beschreibungsmechanismus auch diskrete Plätze und Transitionen erhalten. Im Unterschied zu den diskreten Plätzen wird in kontinuierlichen Plätzen nicht der Markenfluss beobachtet, sondern die kontinuierlich steuerbare Veränderung eines bestimmten Wertes. Die kontinuierlichen Transitionen schalten dabei bei der Erfüllung von allen Schaltbedingungen nicht diskret, sondern mit einer durch eine kontinuierliche Zeitfunktion angegebenen Geschwindigkeit. Diskrete und kontinuierliche Komponenten kommunizieren miteinander nach bestimmten vordefinierten Regeln. Bei der Modellierung von Zusammenhängen zwischen ereignisdiskreten und kontinuierlichen Prozessen und Signalen sind die möglichen Kombinationen in hybriden Petri-Netzen voll ausreichend.

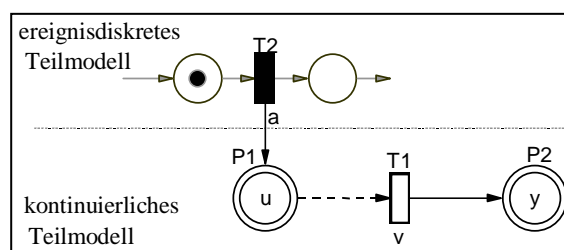


Abbildung 7: Ereignisdiskrete Generierung von Sprüngen einer kontinuierlichen Prozessgröße

- Abbildung 7 [Drath 99] zeigt eine von mehreren Möglichkeiten, diskretes und kontinuierliches Verhalten miteinander zu verbinden. Hier wird ein kontinuierlicher Pro-

zess nach einem zeitdiskreten Ereignis mit der Generierung einer Sprungfunktion gestartet.

2.3 Konzepte der Objekttechnologie und der UML als Standard für die objektorientierte Modellierung

Der Entwurf von verteilten eingebetteten Echtzeitsystemen ist aufgrund der oben genannten Besonderheiten ein komplexer Prozess. Dabei stellt die Objektorientierung ein wichtiges Konzept zur Bewältigung von struktureller Komplexität des zu entwerfenden Systems zur Verfügung. Die objektorientierte Technologie, bei der die Daten und die sie bearbeitenden Funktionen in Objekten zusammengefasst werden, welches zum Schluss zu einem abstrakten Datentyp, einer so genannten Klasse führt, entspricht einerseits besser der realen Welt und verringert andererseits, durch die Zusammenfassung von Daten und Bearbeitungsmethoden für diese Daten, die Komplexität der zu entwerfenden Modelle. Mit Hilfe der objektorientierten Konzepte, wie Kapselung, Vererbung, Polymorphismus und Instanziierung können die Modelle eines zu entwerfenden Systems leichter erstellt, überprüft und verwendet werden. Ein objektorientierter Entwurf besteht aus der Erstellung eines Systems von Objekten, deren Struktur und Verhalten durch ihre Klassenbeschreibung festgelegt wird, und aus der Beschreibung des Kommunikationsmechanismus zwischen den Objekten. Dabei gelten als Aufgaben das Auffinden der geeigneten Objektklassen, das Festlegen der Attribute und Methoden dieser Klassen, das Aufdecken spezieller Beziehungen untereinander, wie z.B. Vererbung (Beziehung: "ist-ein(e)") oder Aggregation (Beziehung: "ist-Teil-von").

Die Anzahl objektorientierter Methoden stieg seit Anfang der achtziger Jahre deutlich an, so dass eine Standardisierung notwendig wurde. Die dabei entstandene Unified Modeling Language (UML) ist eine Standardsprache und Notation für die Entwicklung objektorientierter Systeme. Als Zusammenführung der Object Modeling Technique (OMT [Rumb 91]), des Object-Oriented Software-Engineering (OOSE [Jaco 92]) und der Booch-Methode ([Booc 91]) bietet UML eine Vielzahl von graphikorientierten Darstellungsmöglichkeiten zur Spezifikation, Konstruktion, Visualisierung und Dokumentation objektorientierter Systeme. Da schon eine deutliche Tendenz zur Ersetzung bzw. Ergänzung algebraischer Beschreibungsmittel durch grafische existiert, gewinnen solche grafischen Beschreibungstechniken immer mehr an der Bedeutung.

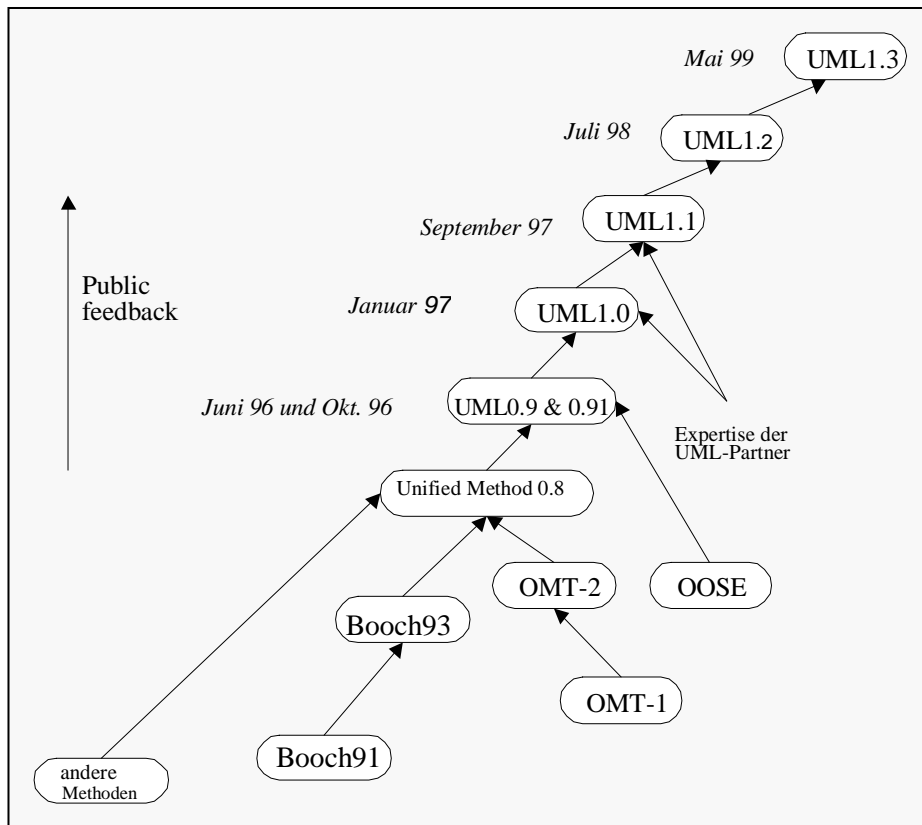


Abbildung 8: Entwicklung der UML nach [Camb 98] mit Erweiterungen

Ein wesentliches Ziel bei der Entwicklung der UML (Abbildung 8) war das Erreichen der Unabhängigkeit von verschiedenen Programmiersprachen, um den Wiederverwendbarkeitsgrad zu erhöhen [Gamb 98]. Die UML stellt eine Sprache und Notation zur Modellierung und nicht eine Methode dar, was die Anwendbarkeit von UML bei einem großen Spektrum von Modellierungsansätzen erlaubt. Die UML kann eine Basis für verschiedene Methoden sein [Burk 97]. Die Architektur der UML besteht aus grafischer Notation und definierendem Metamodell (Modell für Modellierungselemente), welches die Möglichkeit zu anwendungsspezifischen Erweiterungen bietet. Das UML-Metamodell besitzt eine komplexere Struktur – sie besteht aus ca. 90 Metaklassen. Diese werden mit starker Verbundenheit hierarchisch in Paketen organisiert. Die abstrakte Syntax wird durch Klassendiagramme mit anschließender Beschreibung in natürlicher Sprache dargestellt. Die Semantik wird in natürlicher Sprache oder einer Spezifikationssprache beschrieben. Für alle Elemente in der UML sind eine einheitliche Notation und grafische Diagramme zur Darstellung und Kooperation von Elementen vorhanden. Die verschiedenen Diagrammtypen projizieren unterschiedliche Sichten auf das Gesamtmodell. Für die Anwendung der UML in Echtzeit- bzw. Eingebetten Systemen existieren speziell dafür erweiterte Varianten [SeRu 98, SeRu 99, Doug 98, Doug 2000].

2.3.1 Beschreibungselemente der UML

Die Beschreibungselemente der UML bestehen aus Grundelementen, welche Elementen der Objekttechnologie entsprechen, und verschiedenen Diagrammtypen, welche verschiedene Sichten auf ein zu entwerfendes System repräsentieren.

Wichtige Grundkomponenten und Begriffe der UML sind: Klasse, Objekt, Methode, Prozess, Ereignis, Assoziation/Rolle, Vererbung, Polymorphismus. Dabei lassen sich die Grundelemente der UML in zwei Kategorien einordnen: statische und dynamische. Abbildung 9 nach [Burk 97] zeigt die Einteilung verschiedener Elemente unter dem Blickwinkel statischer oder dynamischer Natur.

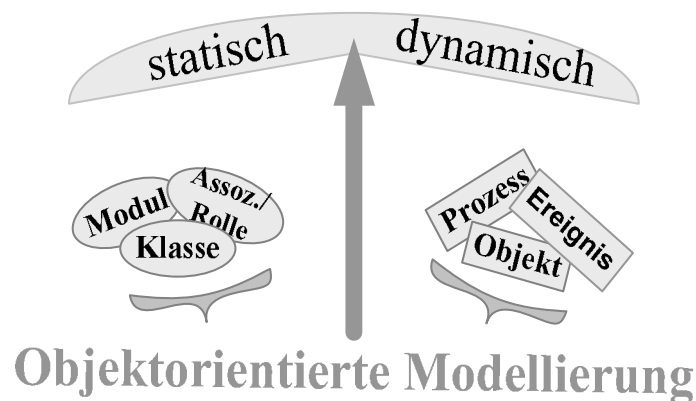


Abbildung 9: Einteilung nach statischen und dynamischen Aspekten

Zu statischen Aspekten gehören im wesentlichen die Klassen und Methoden. Nach [UML-1.3] repräsentiert eine Klasse eine Menge von Objekten mit ähnlicher Struktur, Verhalten und Beziehungen, welche eine Struktur mit Zusammensetzung aus Attributen und Operationen darstellt und die Vererbung spezifiziert. Attribute sind Datenelemente, die in jedem Objekt einer Klasse gleichermaßen enthalten sind und von jedem Objekt mit einem individuellen Wert repräsentiert sind. Operationen sind Dienste, die von einem Objekt aufgerufen werden können und die durch Operationsname, Parameter und Rückgabewert beschrieben werden. Eine Methode, welche die Aufgabe der Beschreibung eines Verhaltens des einzelnen Objekts hat, stellt die Implementierung von einer Operation dar und besteht aus einer Folge von Anweisungen. Zur Beschreibung von Randbedingungen einer Methode kann in der UML seit Version 1.1 die Object Constraints Language (OCL) benutzt werden.

Das Objekt als Repräsentant eines dynamischen Aspektes stellt ein aktives, konkret vorhandenes Modellelement in einem System dar. Es ist ein Exemplar einer Klasse, welches eine bestimmte Anzahl der durch die Klasse definierten Attribute besitzt, deren Werte aber für jedes Objekt individuell sein können. Die in der Klasse festgelegten Zugriffsrechte sind auch für Objekte relevant und unterstützen damit ein wichtiges Konzept der Objektorientierung, nämlich die Kapselung. Ein Objekt stellt eine Instanz einer Klasse dar, dabei stellt ein Prozess eine Instanz einer Methode dar, was aber grafisch in UML nicht direkt unterstützt ist. Ein Prozess wird durch eine bestimmte Art der Aktivierung und relevante Zeitangaben charakteri-

siert. Die Ausführung von Prozessen erfolgt nur unter Einhaltung von Randbedingungen: Vor- und Nachbedingungen. Ereignisse werden in der UML in ihrer Wirkungsweise auf Übergänge von einem Zustand zu einem anderen betrachtet. Eine ausführliche Beschreibung von objektorientierten Komponenten und Begriffen befindet sich in [Booc 91, Booc 94, Meye 90, KePf 90] und speziell zur UML in [UML-1.3, Burk 97, Oest 98, Hitz 99]

Zu der in der UML enthaltenen Vielzahl von Modellelementen und Details gehören auch verschiedene Diagrammtypen. Diese Diagramme repräsentieren verschiedene Aspekte, auch Sichten genannt, auf das Modell des zu erstellenden Systems. Jedes Diagramm beschreibt daher in der Regel nur einen Teil eines Aspekts eines zu entwickelnden Systems. Dabei werden für die konkreten Anwendungsgebiete meistens nur bestimmte Diagrammtypen verwendet und selten ein System parallel durch alle existierenden Diagrammtypen beschrieben. Jedes Diagramm ist öfters nur für bestimmte Phasen des Modellierungsprozesses geeignet.

2.3.2 Diagrammübersicht

Die UML definiert acht Diagrammformen und legt einen Teil der Organisation und die vollständige Repräsentation der Beschreibungselemente in den Diagrammen fest. Manche Beschreibungselemente können in der UML in mehreren verschiedenen Diagrammtypen vorkommen. Die folgenden Darstellungen zu den einzelnen beschriebenen Diagrammen basieren auf [OMG 2004].

In der UML existieren folgende in weiteren Unterkapiteln beschriebene Diagrammtypen.

2.3.2.1 Anwendungsfalldiagramm (auch Use-Case-Modell genannt)

In diesem Diagramm wird auf ziemlich allgemeinem Niveau die Funktionalität des zu entwickelnden Systems in Form von Anwendungsfällen angegeben. Dieses Diagramm spezifiziert eine Schnittstelle zum Anwender: das Systemverhalten, welches das System leisten muss, wird aus der Sicht des Anwenders beschrieben. Das Verhalten des Gesamtsystems wird in einem Modell abgebildet, welches aus allen möglichen Anwendungsfällen besteht. Das Anwendungsfalldiagramm beschreibt dabei die Beziehungen zwischen so genannte Akteuren, den außerhalb des Systems liegenden Kommunikationspartnern, welche Personen oder andere Systemen darstellen und den Anwendungsfällen. Diese Diagramme können auch untereinander gewisse Beziehungen haben und hierarchisch aufgebaut sein. Das in Abbildung 10 gezeigte Beispiel für ein Use-Case-Diagramm, welches in Rahmen eines Projektes TU Ilmenau im Fachgebiet Rechnerarchitektur [DFG-ZB 01, DFG-AB 03, DuHuFeFe 04] entwickelt wurde, beschreibt die zu erbringenden Dienste einer modellierten Steuerung eines Laser-Messtischs. Dieses Beispiel soll einheitlich über die ganze Arbeit zeigen, wie die schon bekannten und die neuen, in dieser Arbeit entwickelten Ansätze beim Entwurf realer Systeme angewendet werden können.

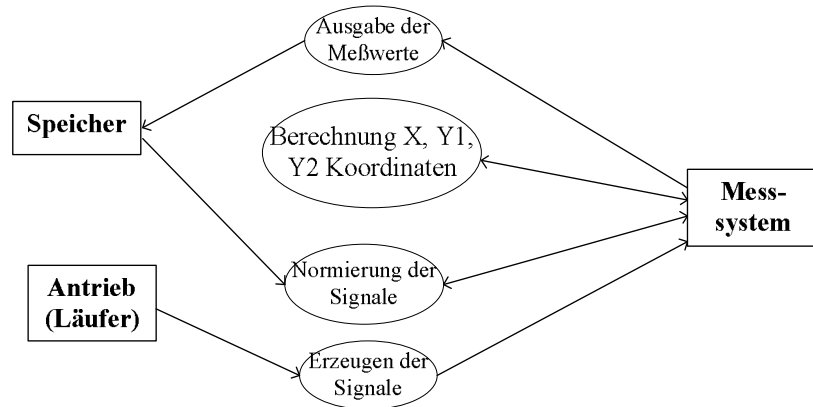


Abbildung 10: Use-Case-Diagramm für ein Lasermesssystem

Dieser Lasermesstisch besitzt ein Präzisions-Positionierungssystem. Es sind dabei zwei Freiheitsgrade zugelassen für Bewegungen in zwei Richtungen in einer Ebene: X- und Y- Richtung. Dabei wird auch eine eventuell mögliche unerwünschte Verdrehung gegenüber dem Koordinatensystem gemessen und ausgeglichen. Dafür erfolgt eine optische Abtastung eines Kreuzrasters mit Hilfe eines speziellen Messkopfes (Abbildung 11).

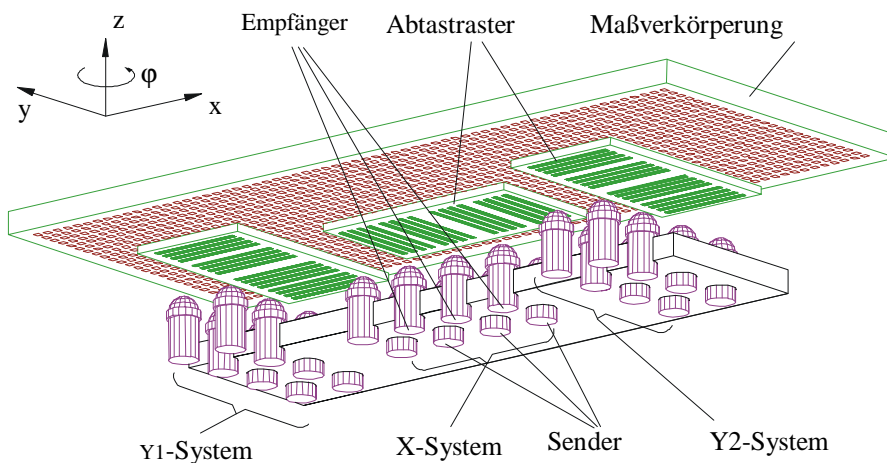


Abbildung 11: Messkopf [DuHu 2001]

Die eigentliche Messwertbildung erfolgt durch die Auswertung von Interferenzmustern. Dabei entstehen als Eingabewerte Sinus-förmige Signale (mit Phasenverschiebung zueinander, deshalb wird im weiteren mit Sinus und Cosinus bezeichnet). Die mögliche Verdrehung des Läufers wird durch die Verwendung zweier Abtastraster für die Y-Richtung festgestellt. Dabei werden folgende Messwerte ermittelt: einmal die Positionsbestimmung für die x-Achse und zweimal die Positionsbestimmung für die y-Achse, durch welche eine eventuelle Abweichung durch Verdrehung festgestellt und korrigiert werden können. Eine nähere Beschreibung zu diesem Messsystem ist in [SaSc 96, DuHu 01, Döri 02] zu finden.

2.3.2.2 Klassendiagramm

Dieses Diagramm beschreibt die statische Struktur des zu entwickelnden Systems in Form von Klassen und deren Beziehungen zueinander. Es visualisiert die Zusammenhänge zwischen den Klassen. Dieses Diagramm enthält statische Strukturen von Klassen mit Vererbungsbeziehungen und Assoziationen und Klasseninhalte mit Attributen und Methoden, die im Klassensymbol nur durch Namen und relevante Parameter angegeben sind und dabei keine Ablaufreihenfolge aufweisen. Zusätzlich können in diesen Diagrammen als Instanziierung der Klasse Objekte gezeigt werden. Eine Klasse kann Assoziationen, die für die Kommunikation zwischen Objekten nötig sind, und Rollen zu einer anderen Klasse besitzen. Die Rollen können im Entwurf als Attribute dieser Klasse erscheinen. Die Rollen geben an, wie viele Objekte an bestimmten Beziehungen beteiligt sind. Gerichtete Assoziationen sind ein spezieller Fall von Assoziationen, welche nur einseitige Beziehungen zwischen Klassen angeben.

Die Klassen werden als Rechtecke mit einer Teilung in drei Bereiche dargestellt. Dabei wird in dem oberen Bereich der Name der Klasse angegeben, im mittleren Bereich werden die Attribute von dieser Klasse aufgelistet und im unteren Bereich stehen die Methoden der Klasse. Die Vererbungsbeziehungen werden durch hohle Pfeile, die auf die Basisklassen zeigen, angegeben. Bestimmte Spezialisierungen können an der Kante vorhanden sein. Ungerichtete Assoziationen werden durch einfache Linien gezeigt, dagegen werden gerichtete Assoziationen durch spezielle Pfeile, z.B. wie im Aggregationsfall, gekennzeichnet. Die Abbildung 12 [Gren 2003] zeigt die statische Struktur von Klassen im modellierten Beispiel des Messtisches.

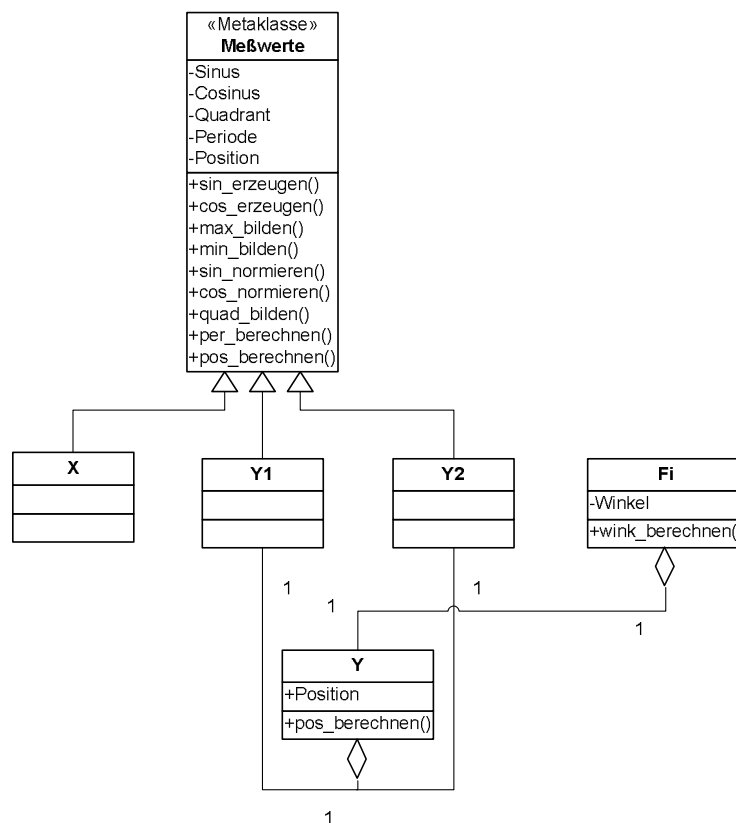


Abbildung 12: Klassendiagramm

2.3.2.3 Zustandsdiagramm

Es gehört zu den Verhaltensdiagrammen. Die Technik des Zustandsdiagramms (der Statecharts) und ihre grafische Darstellung wurden 1987 von David Harel [Hare 87] entworfen. In der UML wurden sie objektorientiert erweitert [UML 2]. Zustandsdiagramme für eine Klasse zeigen die Folge von Zuständen auf, welche ein Objekt dieser Klasse im Laufe seines Lebens (von seiner Erzeugung bis zu seiner Destruktion), bzw. während der Ausführung einer Operation oder Interaktion einnehmen kann. Zustandsdiagramme stellen das dynamische Verhalten von Objekten dar und werden meistens für die Beschreibung des Klassenverhaltens sowie für die Beschreibung von verschiedenen Methoden, Operationen oder zur Modellentwicklung zustandsorientierter technischer Systeme benutzt. In den UML-Zustandsdiagrammen wird ein so genanntes Ereigniskonzept unterstützt.

Ein Zustandsdiagramm stellt einen Graph dar, dessen Knoten den Zuständen, die von Objekten eingenommen werden können, entsprechen und dessen Kanten (oder Transitionen) die möglichen Zustandsübergänge angeben.

Die Objekte befinden sich eine gewisse Zeit lang in einem bestimmten Zustand, bis ein Ereignis einen Zustandsübergang auslöst, welcher dann unmittelbar erfolgt. Den Zuständen können Aktivitäten zugeordnet werden, die ausgeführt werden, solange sich das Objekt im jeweiligen Zustand befindet. Zustände sind hierarchisch verfeinerbar: Ein Zustand kann durch ein weiteres Zustandsdiagramm beschrieben werden. Dabei kann es sich um UND-Verfeinerung (Komposition von zwei oder mehreren parallelen Unterzuständen) bzw. ODER-Verfeinerung (Komposition von sich gegenseitig ausschließenden Unterzuständen) handeln (Abbildung 14,15).

Die Zustände werden durch Rechtecke mit abgerundeten Ecken gekennzeichnet, welche wiederum folgende Teile beinhalten können:

- Name des Zustandes. Zustände ohne Namen sind möglich, dabei anonym und verschieden.
- Innere Transitionen. Die inneren Aktivitäten sind einem Zustand zugeordnet und werden durchgeführt, wenn ein Objekt sich in diesem Zustand befindet. Dabei sind folgende Bezeichnungen vorgesehen: Aktions-Beschriftung/ Aktions-Ausdruck. Dabei bestimmt die Aktions-Beschriftung die Bedingung, unter welcher der Aktions-Ausdruck ausgeführt werden kann.

Folgende vier Beschriftungen sind reserviert:

- „entry“: Die Aktion wird beim Eintreten in den Zustand durchgeführt.
- „exit“: Die Aktion wird beim Verlassen des Zustands durchgeführt.
- „do“: Die Aktion wird so lange ausgeführt, so lange ein Objekt sich in diesem Zustand befindet oder der Vorgang, welcher im Aktions-Ausdruck bestimmt ist, nicht beendet ist.
- „include“: Sie wird für das Identifizieren einer Submaschine benutzt.

In allen anderen Fällen stellt die Aktions-Beschriftung ein Ereignis dar, welches den nachfolgenden Aktions-Ausdruck aktiviert. Allgemein wird folgendes Format für innere Transitionen benutzt:

event-name '(' comma-separated-parameter-list ')' '[' guard-condition ']' '/' action-expression, wobei event-name ein Ereignisname, comma-separated-parameter-list eine Liste von Parametern, guard-condition die Überwachungsbedingung und action-expression ein Aktionsausdruck ist. Bei unterschiedlichen Überwachungsbedingungen kann ein Ereignisname mehrfach vorkommen (Abbildung 13).

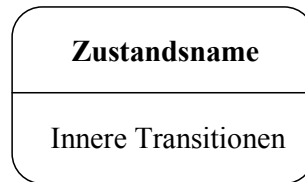


Abbildung 13: Zustandsdarstellung

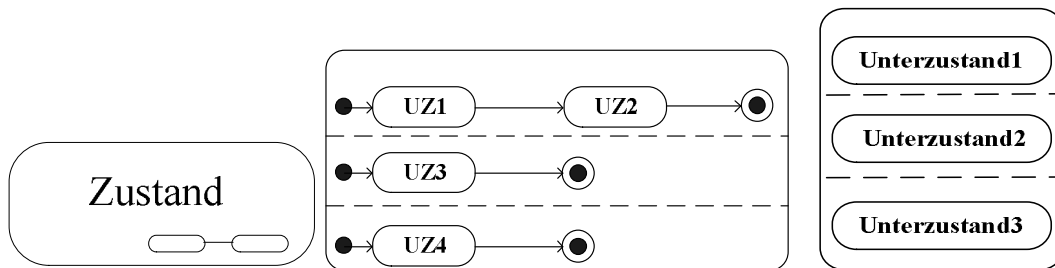


Abbildung 14: Verfeinerter UND-Zustand

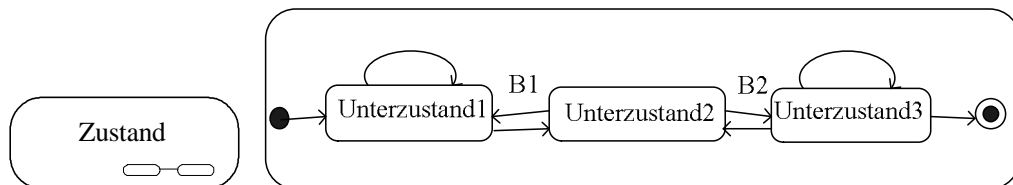


Abbildung 15: Verfeinerter ODER-Zustand

In Abbildung 16 ist ein so genannter Synchronisations-Zustand (Synch-Zustand, S1), welcher die Abhängigkeit einer Transition von einer anderen in einem UND-Verfeinerungszustand steuert, gezeigt.

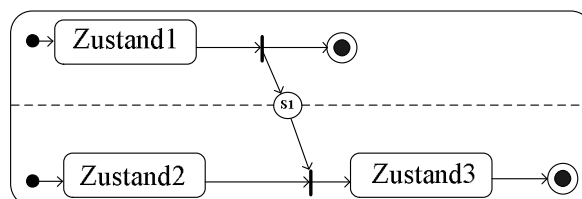


Abbildung 16: Synchronisations-Zustand

Zustandsübergänge beschreiben den Wechsel von einem Zustand in einen Folgezustand unter der Einhaltung von spezifizierten Bedingungen. Zustandsübergänge werden durch verschiedene Ereignisse, z.B. durch bedingungsorientierte Ereignisse, durch Signalereignisse oder durch Zeitereignisse ausgelöst. Die Ereignisse sind unter folgendem Format definiert:

event-name ‘(‘ comma-separated-parameter-list ‘)’, wobei event-name ein Name und comma-separated-parameter-list eine Liste von Parametern, welche mit folgendem Format parameter-name ‘:’ type-expression (Parametername und Ereignistyp) versehen sind, darstellen. Beim Eintreten eines Ereignisses, welchem im aktuellen Zustand keiner Transition zugeordnet ist, geht dieses verloren.

Die Transitionen verbinden Zustände mit ihren Folgezuständen und werden mit Hilfe durchgezogener Pfeile mit Beschriftungen in folgendem Format notiert:

event-signature ‘[‘ guard-condition ‘]’ ‘/’ action-expression (Ereignissignatur, Überwachungsbedingung, auszuführenden Aktionen), dargestellt.

Es gibt einfache und komplexe Zustandsübergänge. Einfache Transitionen verbinden zwei Zustände, komplexe können mehrere Zustände mit einander mit Hilfe so genannter Synchronisationselemente verbinden. (Abbildung 17).

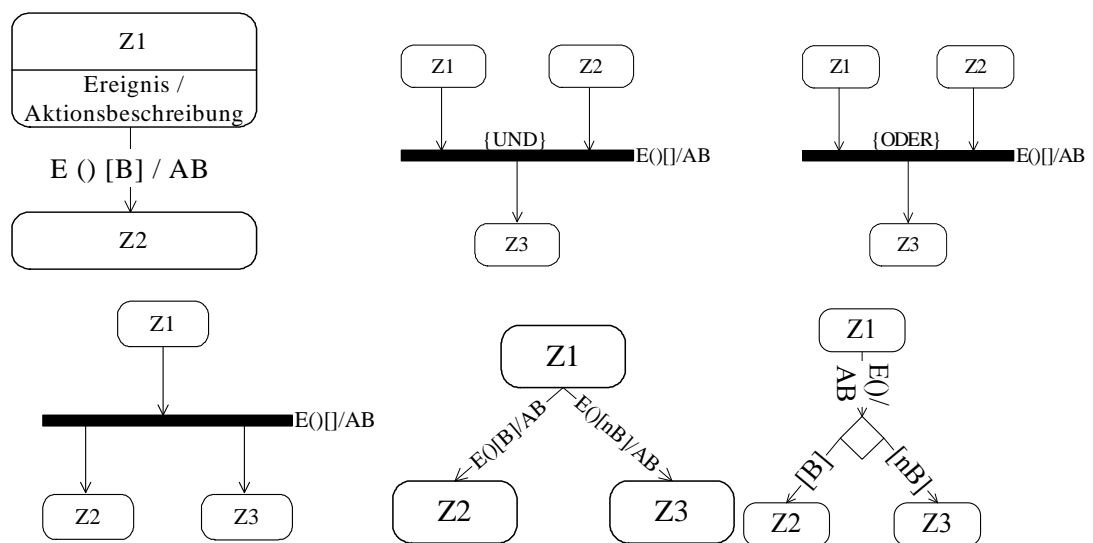


Abbildung 17: Einfacher Zustandsübergang, UND- bzw. ODER-Synchronisationen, parallele Verzweigung und Entscheidungsvarianten.

In den UML-Diagrammen ist eine optionale Transitionszeit vorgesehen. Es ist aber nicht eindeutig spezifiziert, wo die Transitionszeit einzugeben ist.

Die auf den klassischen Zustandsdiagrammen von D. Harel basierenden Zustandsdiagramme der UML weisen folgende Erweiterungen auf: In Zustandsdiagrammen der UML besitzen die Ereignisse Parameter, anstatt nur Signale darzustellen. Dabei findet keine Ereigniskonjunktion statt, sondern das Ereigniskonzept wird als Bearbeitung der einzelnen Ereignisse definiert. Es wird synchrone Kommunikation zwischen Zustandsmaschinen unterstützt. Als neues Ele-

ment wird eine dynamische Entscheidungsstelle eingeführt. Bei klassischen Zustandsdiagrammen benötigen die Transitionen keine Zeit für das Schalten, bei objektorientierten Zustandsdiagrammen der UML ist es möglich, dass diese Zeit benötigen.

In Abbildung 18 [Baru 2002] ist ein Ausschnitt aus dem Zustandsdiagramm des modellierten Messsystems gezeigt. In diesem Diagramm sind einfache, verfeinerte, zusammengesetzte Zustände und Synchronisations-Zustände dargestellt. Die parallele Berechnung der Y1- und Y2- Koordinaten ist mit Hilfe der UND-Verfeinerung realisiert.

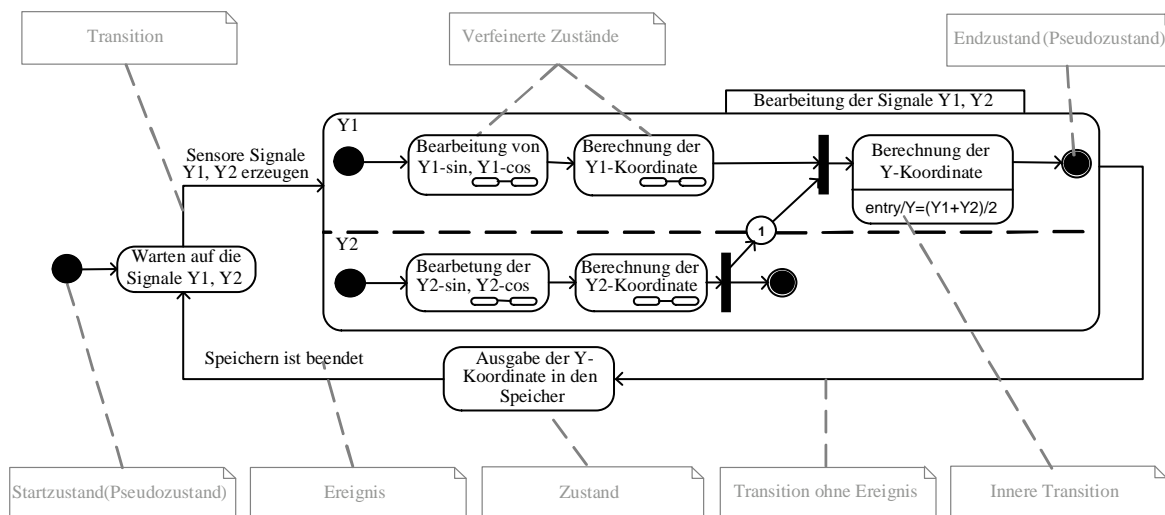


Abbildung 18: Zustandsdiagramm für ein Messsystem

2.3.2.4 Aktivitätsdiagramm.

Es gehört zu den Verhaltensdiagrammen. Die Aktivitätsdiagramme basieren auf den Ereignisdiagrammen von Martin und Odell. Sie sind in der UML als Spezialisierung der Zustandsdiagramme definiert und bestehen aus Zuständen und Zustandsübergängen. Ein variierbares Abstraktionsniveau in der Detaillierung von Aktivitätsdiagrammen bietet zahlreiche Verwendungsmöglichkeiten bei der Modellierung. Ein Aktivitätsdiagramm spezifiziert eine Ablaufreihenfolge von Prozessen, welche sein können: Geschäftsprozesse, Rechenprozesse, Realisierungen von einzelnen Operationen, Algorithmusausführungen, Beschreibung von einem oder dem Zusammenspiel von mehreren Anwendungsfällen [Burk 97, Mühl 98, Öste 98]. Wie ein Zustandsdiagramm, beginnt ein Aktivitätsdiagramm mit dem Startzustand und endet mit einem oder mehreren Endzuständen.

Die Zustände repräsentieren die Ausführung von Aktivitäten eines oder mehrerer Objekte, die Zustandsübergänge werden nach dem Abschluss einer solchen Operation aktiviert (Abbildung 19). Dabei kommen externe Ereignisse in Aktivitätsdiagrammen selten vor. Die Zustandsübergänge (Transitionen) können jedoch mit einer Bedingung versehen werden. Dabei wird ein solcher Zustandsübergang nur ausgeführt, wenn eine vorstehende Aktivität abgeschlossen und die Bedingung erfüllt ist. Diese meistens mit logischen Ausdrücken formulierten Bedingungen sind meist als Alternativen im Fall von mehreren fortführenden Transitionen notiert. Wie bei den Zustandsdiagrammen werden Transitionen in einfache und komplexe

unterschieden. Eine Variante komplexer Transitionen stellt das Aufteilen des Informationsflusses in mehrere parallele Prozesse dar. Dabei werden diese parallelen Prozesse explizit beschrieben. Die andere Variante komplexer Transitionen stellt das Aufteilen in mehrere Alternativen dar. Diese beiden Transitionsformen sind in Abbildung dargestellt.

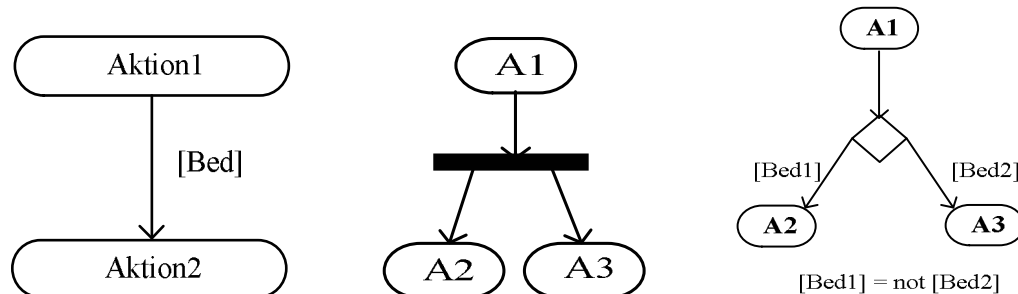


Abbildung 19: Elemente des Aktivitätsdiagrammes: Zustände und Zustandsübergänge, Parallelitäts- und Entscheidungskonzepte

Die Möglichkeit der Zuordnung einzelner Aktivitäten zu Objekten, in deren Zuständigkeitsbereich sie sich befinden, wird in Aktivitätsdiagrammen durch so genannte „Schwimmbahnen“ realisiert werden. Bei senkrechter Darstellung von Aktivitätsdiagrammen können zusätzliche Zeitbedingungen für Transitionen definiert werden.

In der UML wird ein Aktivitätsdiagramm einer Klasse, einer Methode oder einem Use Case zugeordnet und beschreibt damit das dynamische Verhalten des entsprechenden Elements. Wie schon gesagt, das Ziel des Aktivitätsdiagramm ist die Beschreibung einen Ablaufprozesses. In Abbildung 20 [Baru 2002] ist ein einfaches Aktivitätsdiagramm (Auszug) für die Modellierung von Ablaufprozessen in der Steuerung des Laser-Messtisches dargestellt.

An diesem überschaubaren Beispiel kann man die Funktionsweise von Aktivitätsdiagrammen nachvollziehen. Die parallel laufenden Berechnungen der Y1- und Y2-Koordinaten werden mit Hilfe so genannter Splitting- und Synchronisationslinien oder Balken dargestellt, bei denen eine Transition zuerst auf mehrere Transitionen gesplittet wird und danach wieder zu einer Transition zusammengefasst wird.

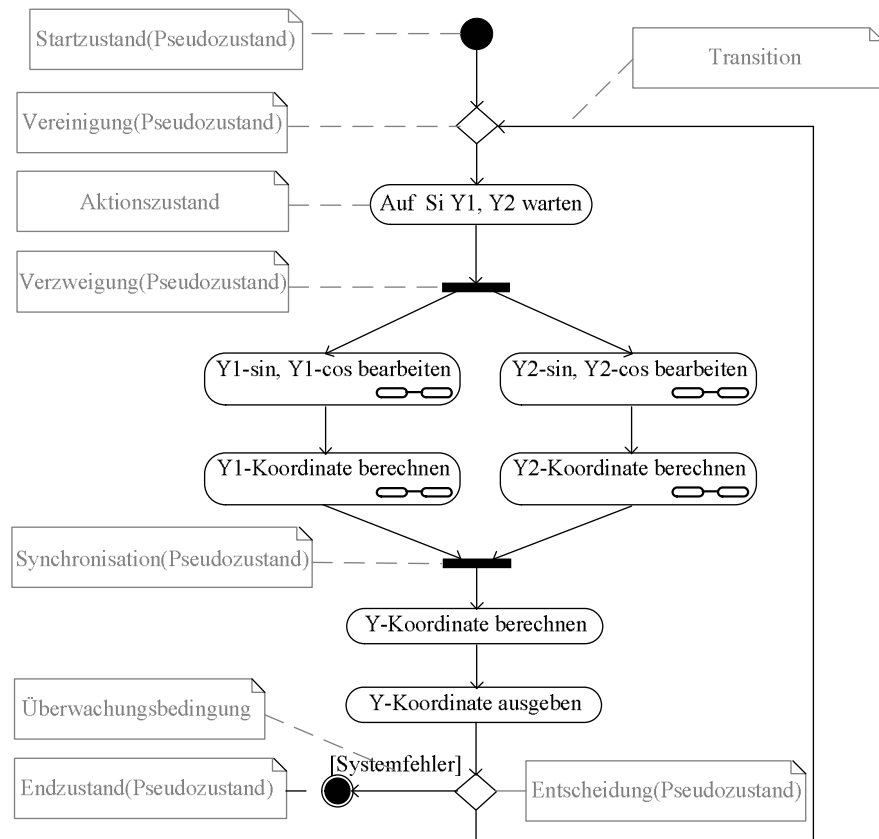


Abbildung 20: Aktivitätsdiagramm

2.3.2.5 Sequenzdiagramm

Es gehört zu den Verhaltensdiagrammen. Die Sequenzdiagramme (SD) zeigen den Austausch von Nachrichten zwischen Objekten und deren zeitlichen Ablauf. Die SD's wurden von den Object Message Sequence Charts (OMSC) [ITU-TS 99] und diese wiederum aus dem Standard MSC-96 abgeleitet. Der grundsätzliche Aufbau und die grafische Anordnung sind dabei erhalten geblieben.

Generell beschreiben Sequenzdiagramme ein Verhalten, bei dem die Interaktion in Form von Nachrichten im Vordergrund steht. Der Nachrichtenaustausch zwischen den verschiedenen Objekten (Instanzen) hat meistens asynchronen Charakter. Was diese Objekte konkret darstellen, ist dabei von untergeordneter Bedeutung. Sie können Softwareprozesse, Hardware oder ähnliches sein. Der Ablauf von Nachrichten wird in Sequenzdiagrammen als ein bestimmtes Szenario dargestellt. In einem Sequenzdiagramm werden zwei Darstellungsdimensionen unterschieden: in horizontaler Richtung werden die Objekte dargestellt und vertikal werden zeitliche Abläufe gezeigt.

Die Objekte in Sequenzdiagrammen werden durch so genannte Lebenslinien repräsentiert, welche als senkrechte gestrichelte Linien ein passives Objekt beschreiben und als mit Aktivierungsbalken versehene Linien ein aktives, an einem Prozess beteiligtes Objekt modellieren. Dabei befindet sich das Objektsymbol an dem oberen Ende dieser Linie und ist als ein Rechteck dargestellt. In Sequenzdiagrammen können Erzeugung und Zerstörung von Objekten durch Instanziierungsnachrichten in dem Fall der Erzeugung und durch Ende der Lebenslinie

der Objekte in einem Kreuz im Fall der Zerstörung gezeigt werden (Abbildung 21). Bei Selbstinteraktion verschickt ein Objekt sich selbst eine Nachricht.

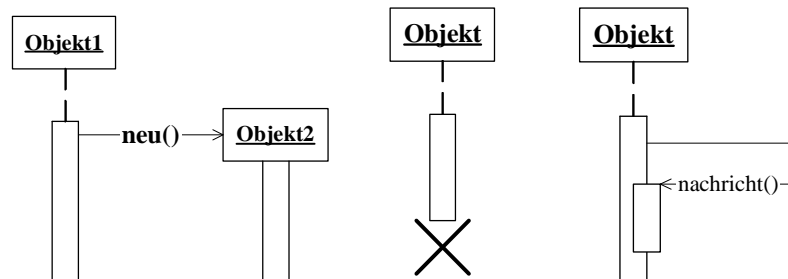


Abbildung 21: Objekterzeugung, Objektzerstörung, Selbstinteraktion

Die Kommunikation zwischen den Instanzen erfolgt durch Nachrichten. Diese bestehen aus einem horizontalen Strich, und ein Pfeil zeigt die Richtung vom Sender zum Empfänger an. Zusätzlich befindet sich der Name der Nachricht direkt an der Linie, gefolgt von einem optionalen Parameter.

Die Sequenzdiagramme unterstützen verschiedene Nachrichtentypen: Synchronen Nachrichten werden als Prozeduraufruf interpretiert. Beim Senden einer solchen Nachricht wird das sendende Objekt auf die Antwort vom Empfänger warten, bevor es mit weiteren Interaktionen anfangen kann. Bei asynchronen Nachrichten interagiert das sendende Objekt weiter, ohne auf die Antwort vom Empfänger zu warten. Weiterhin gibt es Methodenaufrufe und Methodennrückgaben (Abbildung 22).

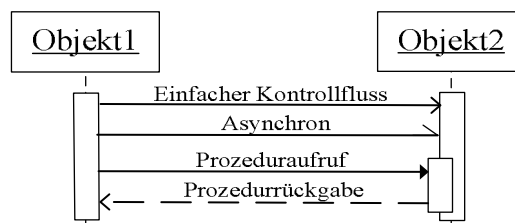


Abbildung 22: Verschiedene Nachrichtentypen in Sequenzdiagrammen

Es sind in Sequenzdiagrammen auch Alternativen und Parallelitäten möglich. Dabei beginnen die zu versendenden Nachrichten mit zusätzlichen Bedingungen in einem Punkt. Dann verzweigen sie sich. Dabei werden bei gleichzeitiger Ausführbarkeit von Bedingungen die Nachrichten parallel gesendet. Wenn die Bedingungen sich paarweise ausschließen, werden die Nachrichten alternativ verschickt. Iterationen von einer einzelnen Nachricht werden mit einem führenden Stern und zwei parallelen senkrechten Strichen vor dem Namen gekennzeichnet (Abbildung 23).

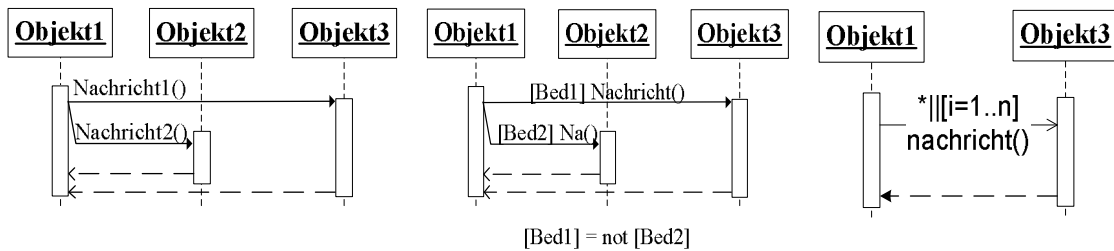


Abbildung 23: Parallelität, Entscheidung, parallele Iteration

In Sequenzdiagrammen lässt sich eine Reihe von Zeitbedingungen definieren. Z.B. können die Nachrichten mit den Zeitbedingungen „nachricht.sendTime“ oder „nachricht.receiveTime“ versehen werden. Der Anwender kann selbst Zeitvorgaben festlegen, welche als Zeitbedingungen am Rande des Sequenzdiagrammes oder direkt neben den Lebenslinien von Objekten notiert werden können.

Zur Veranschaulichung der oben genannten Elemente ist in Abbildung 24 [Baru 2002] ein Ausschnitt aus dem Sequenzdiagramm des Beispiels Messsystem gezeigt.

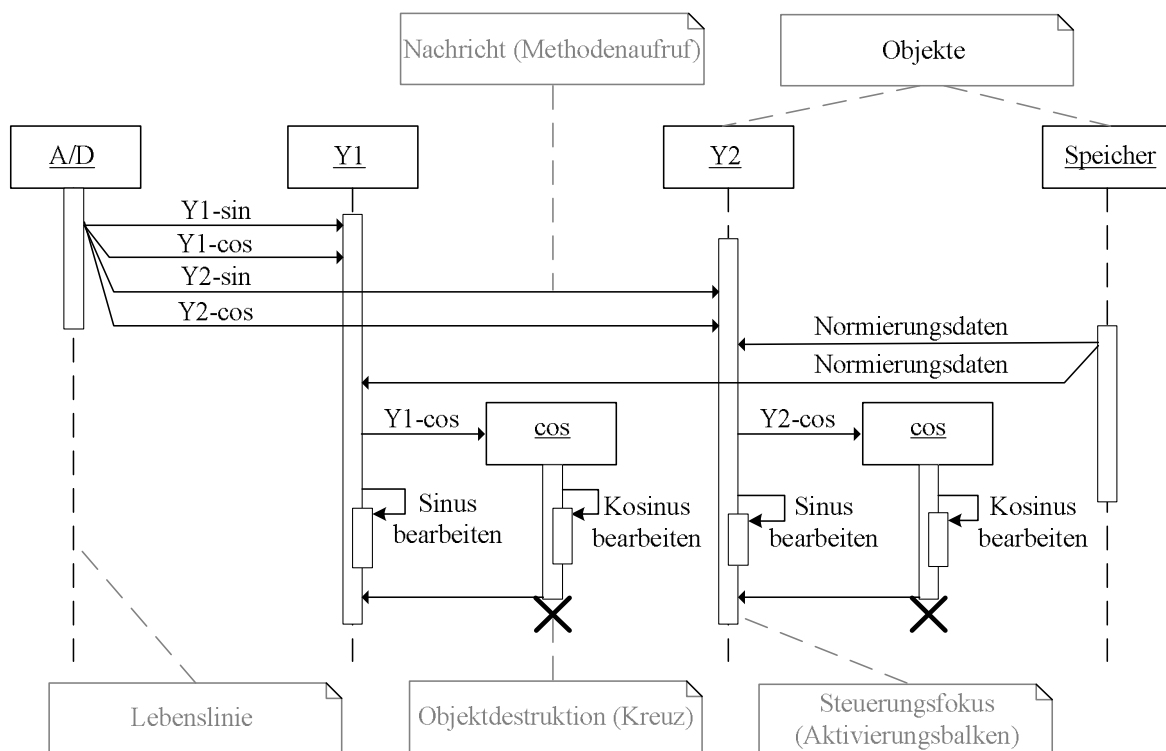


Abbildung 24: Sequenzdiagramm des Beispiels Messsystem

2.3.2.6 Kollaborationsdiagramm

Es gehört zu den Verhaltensdiagrammen. Es zeigt die Interaktionen zwischen Objekten. Dieses Diagramm hat gewisse Ähnlichkeiten mit dem Sequenzdiagramm und ermöglicht die Modellierung praktisch der gleichen Aspekte des Systemverhaltens. Dabei werden aber auch die Assoziations- und Rollen-Beziehungen zwischen Objekten verdeutlicht. Die zeitlichen As-

pekte der Kommunikation zwischen Objekten ergeben sich aus der angegebenen Nachrichtennummerierung. Trotz kompakter Darstellung sind diese Diagramme wegen der stark strukturierten Nummerierung wenig übersichtlich. Im angegebenen Beispiel (Abbildung 25) des Kollaborationsdiagrammes für das oben beschriebene Messsystem werden die Assoziationslinien, die mit Nachrichten beschriftet sind, zwischen Objekten gezeichnet. Dabei zeigt ein kleiner Pfeil die Richtung der Nachrichtenübermittlung zwischen Empfänger und Sender an.

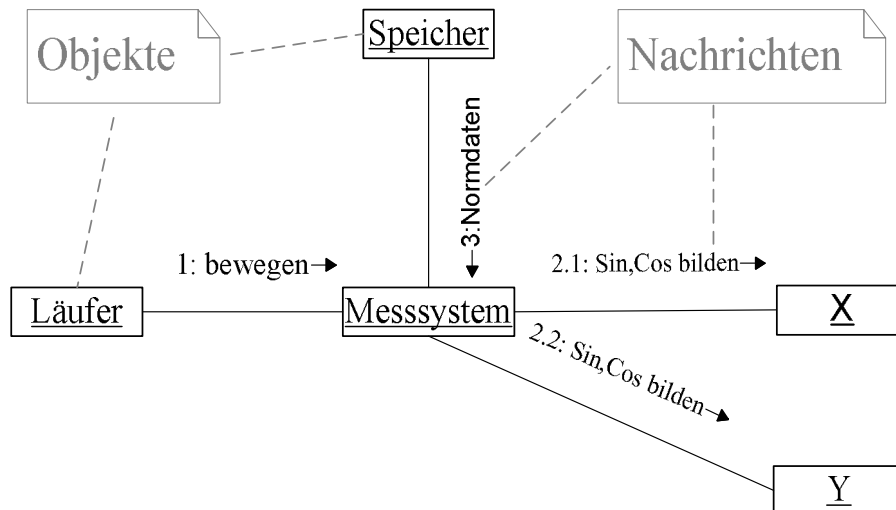


Abbildung 25: Kollaborationsdiagramm für das Messsystem

2.3.2.7 Implementierungsdiagramme

Die dienen zur Darstellung von Implementierungsaspekten und beinhalten so genannte Komponentendiagramme. Sie zeigen die Implementierungsabhängigkeiten von Modellkomponenten untereinander, innerhalb derer sich Klassen, Objekte, Prozesse und andere untergeordnete Module befinden können. Weiterhin gehören dazu die Verteilungsdiagramme. Sie zeigen Komponenten, Knoten und ihre Beziehungen, die durch verschiedene Linien dargestellt sind: Einfache Linien dienen zur Repräsentation von Kommunikationen zwischen Knoten, und gestrichelte Linien dienen zur Darstellung von Abhängigkeiten, z.B. Zugriffen auf ein anderen Knoten. Meist werden Komponentendiagramme und Verteilungsdiagramme zusammen verwendet, um verschiedene Komponenten zu Knoten zuordnen zu können. Ein Beispiel für zusammengesetzte Komponenten und das Verteilungsdiagramm ist in Abbildung 26 [Gren 2003] dargestellt.

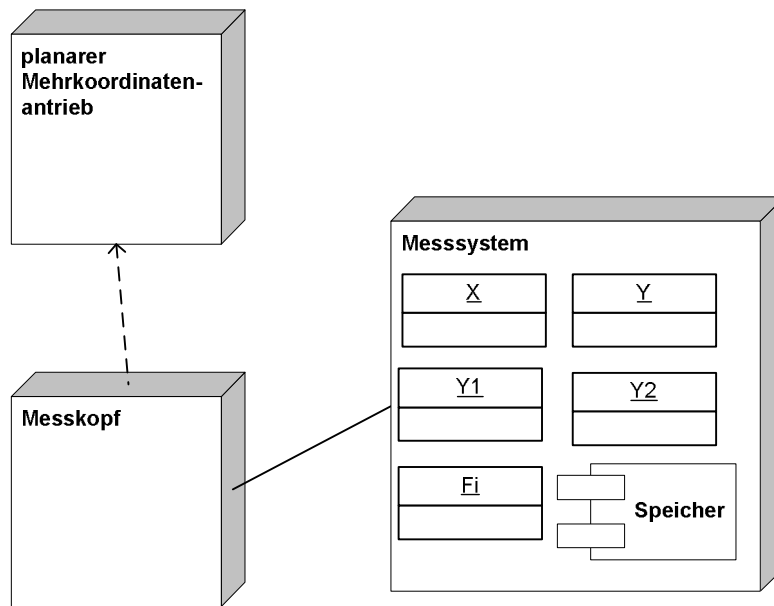


Abbildung 26: Komponenten- und Verteilungsdiagramm

2.4 Höhere Petri-Netze als Beschreibungsmittel

Höhere oder Gefärbte Petri-Netze (Colored Petri Net oder CPN) als eine Variante davon stellen eine Erweiterung von klassischen Petri-Netzen dar, welche durch Einführung von unterscheidbaren, z.B. gefärbten, Marken entsteht. Im Vordergrund steht dabei die kompaktere Darstellung von ähnlichen, sich oft wiederholenden Teilen. Diese treten typischerweise in parallel laufenden Prozessen auf. So werden die möglichen Zustände nicht nur durch die Plätze sondern auch durch die möglichen Farben in den Plätzen repräsentiert.

2.4.1 Grundlegende Definitionen

Da in der Literatur keine einheitliche Definition von gefärbten Petri-Netzen existiert, wird hier auf eine formale Definition für ein möglichst allgemeines gefärbtes Petri-Netz eingegangen [Roky 92, Jens 92, Knor 95].

Das Tupel $CN = (P, T, F, V, K, C, m_{co}, \zeta)$ ist ein gefärbtes Petri-Netz, mit:

1. P Menge aller Plätze
2. T Menge aller Transitionen
3. F Flussrelation: Menge aller Kanten, wobei F die Vereinigung aus Vor- und Nachkanten ist, so dass gilt: $F \subseteq (P \times T) \cup (T \times P)$
4. V Vielfachheit der gefärbten Kanten; Abbildung der Kanten auf eine Menge, bestehend aus Kantengewichten: $P \times T \rightarrow \text{BOOL}(C) \cup T \times P \rightarrow C_T(C)$
5. K Kapazitätsfunktion K für Plätze: $P \rightarrow C_T(C)$
6. C Menge der Farben
7. m_{co} Anfangsmarkierung: $P \rightarrow C_T(C)$

8. ζ Farbkonvertierungsfunktion: $\zeta = \cup S(t)$ für alle $t \in T$ mit $S(t) = \cup S_i(t)$ oder leere Menge und $S_i(t) = (p,t) \rightarrow BOOL(C)$ und $(t,p) \rightarrow C_T(C)$

Dabei ist $C_T(C)$ eine Menge, in der jedes Element aus C höchstens nur einmal vorkommen kann und zwar in Verbindung mit einer natürlichen Zahl als Faktor. Die Menge $BOOL(C)$ stellt eine Menge von Booleschen Ausdrücken mit logischem „und“ und „oder“ über den mit Faktoren versehenen Elementen aus C dar, wobei diese auch geklammert sein können.

Zu den Mengen von Plätzen und Transitionen, welche wie bei normalen Petri-Netzen erhalten bleiben, kommen als Erweiterung noch Mengen von Marken verschiedener Typen, welche durch Farben (C) unterschieden werden. Die Menge C besteht aus allen im Netz auftretenden Farben.

Die Kapazitätsfunktion für die Plätze wird wegen dem Vorkommen verschiedener Markentypen erweitert. Es wird das Auftreten von Marken verschiedener Farben im Platz einzeln begrenzt sowie die maximale Summe von allen Marken angegeben. Wenn $m(p,c)$ die Anzahl von Marken der Farbe „c“ im Platz p ist, dann gilt

$$\forall c \in C, m(c, p) \leq K(p, c) \text{ und} \\ \sum_{c \in C} m(p, c) \leq K(p, \{gesamt\}) \text{ mit } p \in P$$

Die Kantengewichte der Vorkanten von Transitionen stellen einen Booleschen Ausdruck (ohne Negation) aus einer Teilmenge der mit einem Faktor versehenen Farben dar. Die Kantengewichte der Nachkanten von Transitionen sind eine Teilmenge der mit einem Faktor versehenen Farben.

Anstelle der eben beschriebenen Kantengewichte können so genannte Farbkonvertierungsfunktionen genutzt werden. Diese gelten lokal im Vor- und Nachbereich einer Transition und enthalten verschiedene Schaltmodi (Menge $S(t)$), die untereinander mit logischem „oder“ verknüpft sind. Jeder Schaltmodus ($S_i(t)$) definiert für alle Vor- und Nachkanten der Transition eine Kombination von Kantengewichten in der oben beschriebenen Form. Die übliche Weise für die Angabe von Farbkonvertierungsfunktionen sind Tabellen, welche genauso viele Spalten haben, wie die Transition Vor- und Nachkanten hat, und für jeden Schaltmodus eine Zeile besitzen. In den Zeilen stehen entsprechend die Kantengewichte für die Vor- und Nachkanten.

Für die beschriebenen Gefärbten Petri-Netze gilt als Schaltregel:

- Eine Transition t ist dabei schaltfähig, wenn in allen Vorplätzen von t farbrichtig ausreichend Marken vorhanden sind um beim Einsetzen in den Booleschen Ausdrücken aller zu t gehörenden Vorkanten den Wahrheitswert „1“ zu erzeugen. Weiterhin ist notwendig, dass die in allen Nachplätzen von t existierende farbrichtige Differenz zwischen Kapazität und vorhandener Markierung nicht kleiner als die Kantengewichte der Nachkanten ist. Dabei muss noch die Einhaltung der oben genannten Gesamtkapazität eines Platzes gewährleistet sein.
- Beim Schalten wird in den Vorplätzen von t farbrichtig entsprechend der notwendigen Einsetzung in den Kantengewichten subtrahiert und in den Nachplätzen von t addiert.

- „Farbbrichtig“ heißt in dem Zusammenhang, dass sowohl in der Markierung, der Kapazität und den Kantengewichten Terme mit der gleichen Farbe betrachtet werden.

Eine formale Definition zur Schaltregel enthält [Knor 95].

2.4.2 Zeitbewertete Petri-Netze

Um zeitkritische Systeme, z.B. Echtzeitsysteme, mit Hilfe von Petri-Netzen modellieren zu können, wurde ein weiterer Parameter, die Zeit, in Petri-Netze integriert. Ohne diese ist keine auf die Realität von solchen Systemen bezogene Analyse und Verifikation möglich. Es sind so genannte zeitbehafte oder einfach Zeit-Petri-Netze entstanden [Ramc 76, Merl 74, Merl 76, Quäc 91]. Es werden im folgenden nur die im weiteren verwendeten Zeitbewertungsmöglichkeiten betrachtet. Dabei werden als externe Bedingungen von Transitionen Zeitfunktionen, welche gesondert von anderen Bedingungen behandelt werden, hinzugefügt. Es werden drei der am häufigsten benutzbaren Arten von Funktionen für die Zeiten definiert:

- Zeitverzögerung t_v : Dieser Parameter bestimmt die Zeit für ein verzögertes Schalten nach Schaltfähigkeit. Erst nach dieser Zeit kann die Transition in den Vor- und Nachplätzen schalten. Wenn die Bedingungen für Schaltfähigkeit während dieser Zeit nicht erfüllbar sind, wird die Zeit zurückgesetzt.
- Zeitdauer t_d : Hier fängt die Transition sofort zu schalten an: Die Markierung der Vorplätze wird sofort geändert, aber in den Nachplätzen erscheint die Änderung der Markierung erst nach dem Ablauf der angegebenen Zeitdauer. Bei nicht mehr erfüllter Nachbedingung nach Ablauf der Zeitdauer wird die Veränderung der Markierung der Nachplätze zur frühestmöglichen Zeit erfolgen (bei der wieder die entsprechende Schaltfähigkeit der Nachplätze existiert). Das Schalten der Nachbedingung beendet das Schalten dieser Transition.
- Zeitintervall t_i : Hier wird jeder Transition ein Zeitintervall (t_{\min}, t_{\max}) zugeordnet, in dem deren Zeitverzögerung liegt, wobei die für einen konkreten Schaltfall entstehende Verzögerung nicht näher spezifiziert ist. Bei Verlust der Schaltfähigkeit vor dem Schalten wird die Zeit zurückgesetzt, so dass bei der nächsten Wiederherstellung der Schaltfähigkeit die im Intervall angegebene Zeit von vorn läuft.

Diese Parameter können sowohl als Konstanten als auch als Funktionen erscheinen. Da sie nur zusätzliche Bedingungen an Transitionen darstellen, sind sie in klassischen und gefärbten Netzen möglich. In den oben beschriebenen CPN sind die Zeitparameter eventuell noch von der Farbe bzw. dem Schaltmodus der Farbkonvertierungsfunktion abhängig.

Da für die Verifikationsverfahren im Verlauf dieser Arbeit die gefärbten Zeitintervall-Petri-Netze von Interesse sind, wird hier etwas näher auf diesen Typ von zeitbewerteten Petri-Netzen eingegangen.

Dabei ist ein Tupel $CTN = (P, T, F, V, K, C, m_{co}, \zeta, I, f_i)$ ein gefärbtes Zeitintervall-Petri-Netz, wenn:

1. $CN = (P, T, F, V, K, C, m_{co}, \zeta)$ ein gefärbtes Petri-Netz ist
2. I : Menge von Zeitintervallen mit Elementen (t_{\min}, t_{\max})
3. $f_i = S \rightarrow I$ schaltmodusabhängige Zeitintervall-Bewertung der Transitionen

In der Abbildung 27 [Glas 2002] ist ein gefärbtes Zeitintervall-Petri-Netz dargestellt. Es wird jeder Transition in jedem Schaltmodus noch zusätzlich ein Zeitintervall zugeordnet, welches beim Schalten in dem entsprechenden Modus gilt.

Auf Eigenschaften von gefärbten Zeitintervall-Petri-Netzen, wie Erreichbarkeit, Lebendigkeit, Konfliktfreiheit, Beschränktheit und Sicherheit wird bei den Verifikationsmethoden (Kap. 2.6.) eingegangen.

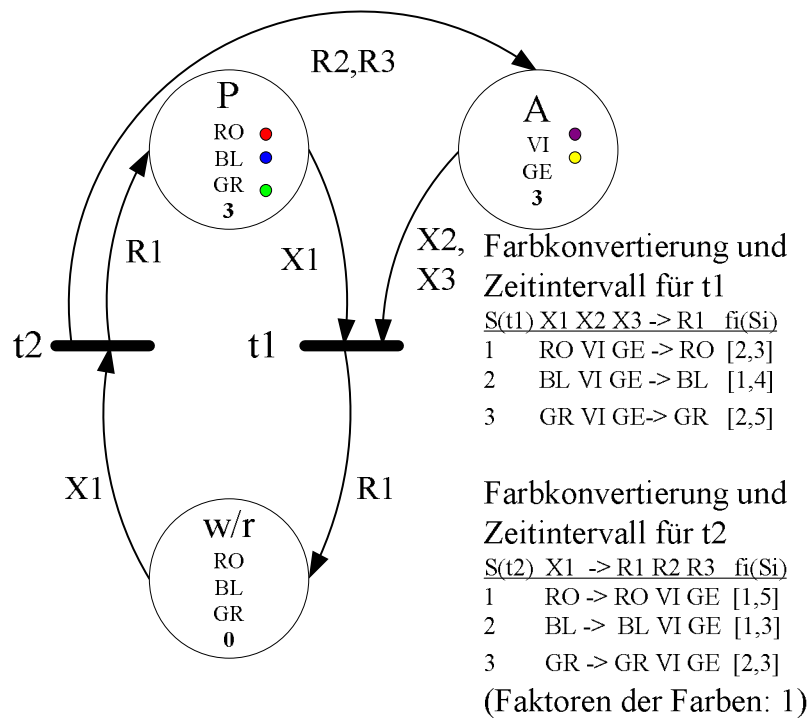
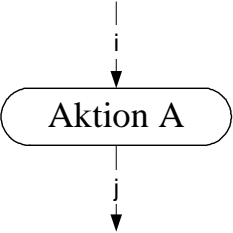
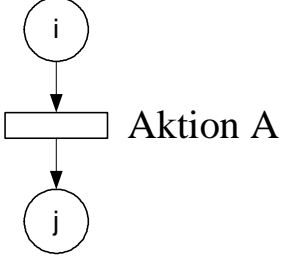
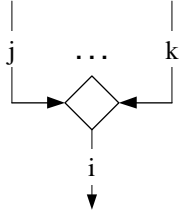
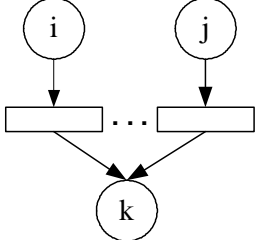
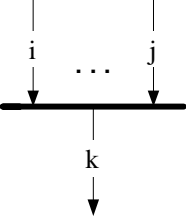
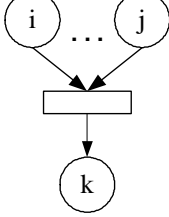
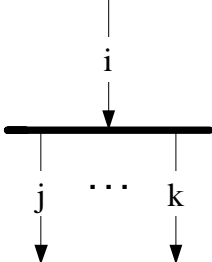
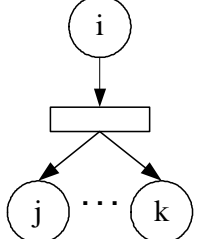
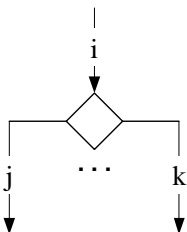
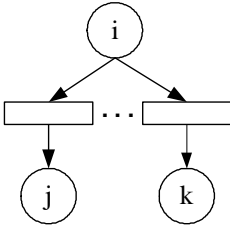




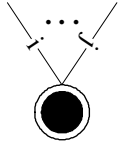

Abbildung 27: Gefärbtes Zeitintervall-Petri-Netz

2.5 Algorithmen zur Überführung von UML-Diagrammen in Petri-Netze

Die Algorithmen zur Transformation von Aktivitätsdiagrammen der UML sind ausführlich in [Mühl 98] dargestellt und weiter in [Rang 2000] ausgearbeitet worden. Da die Aktivitätsdiagramme in der grafischen Darstellung und der Semantik Analogien zu Petri-Netzen aufweisen, kann man sie relativ einfach in diese transformieren. Dazu wird eine Zerlegung in einzelne Bausteine des gesamten Aktivitätsdiagrammes benutzt und diese werden in adäquate Petri-Netz-Bausteine überführt. Nach dieser Prozedur werden zum Schluss gleichnamige Plätze und Transitionen verschmolzen, was zur Entstehung des gesamten Petri-Netzes führt. In der Tabelle 2 [Rang 2000] sind die Überführungen der einzelnen Elemente des Aktivitätsdiagrammes in Petri-Netz-Elemente [Mühl 98] dargestellt.


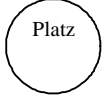
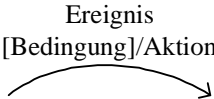
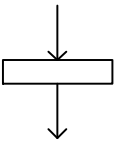
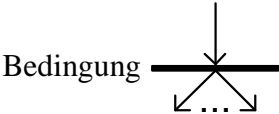
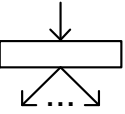
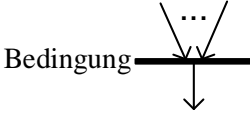
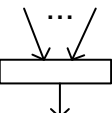
Tabelle 2: Aktivitätsdiagramm- und Petri-Netz-Bausteine

Element	Aktivitätsdiagramm-Baustein	Petri-Netz-Baustein
Aktion		
Vereinigung		
Synchronisation		
Verzweigung		
Entscheidung		
Startzustand		

Endzustand		
------------	---	---

Die Transformationsalgorithmen von Statecharts in Petri-Netze sind in [Abel 87, Fric 96] beschrieben und wurden in [Orec 00] für die Transformation von Zustandsdiagrammen der UML in Petri-Netze angepasst. Dabei wurde auch ein Konzept zur Überführung von Hierarchie in Zustandsdiagrammen in Petri-Netze ausgearbeitet. Die Vorgehensweise ist wie bei den Aktivitätsdiagrammen: Erst wird das Zustandsdiagramm in einzelne Bausteine zerlegt, dann werden diese in Bausteine von Petri-Netzen überführt und zum Schluss erfolgt die Zusammenfassung einzelnen Bausteine zum gesamten Petri-Netz. In der Tabelle 3 sind die Überführungen wesentlicher Bausteine des Zustandsdiagrammes in Petri-Netz-Bausteine [Orec 00] dargestellt.

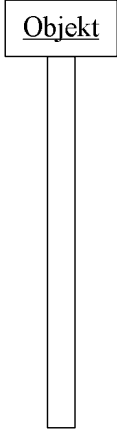
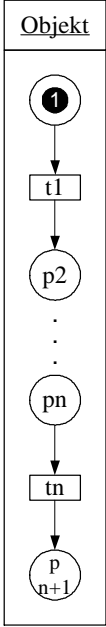
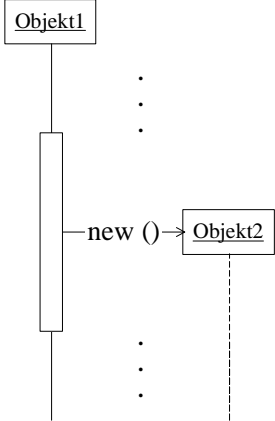
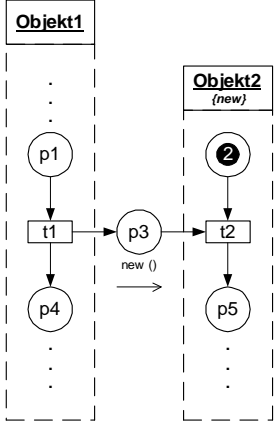
Tabelle 3: Zustandsdiagramm- und Petri-Netz-Bausteine

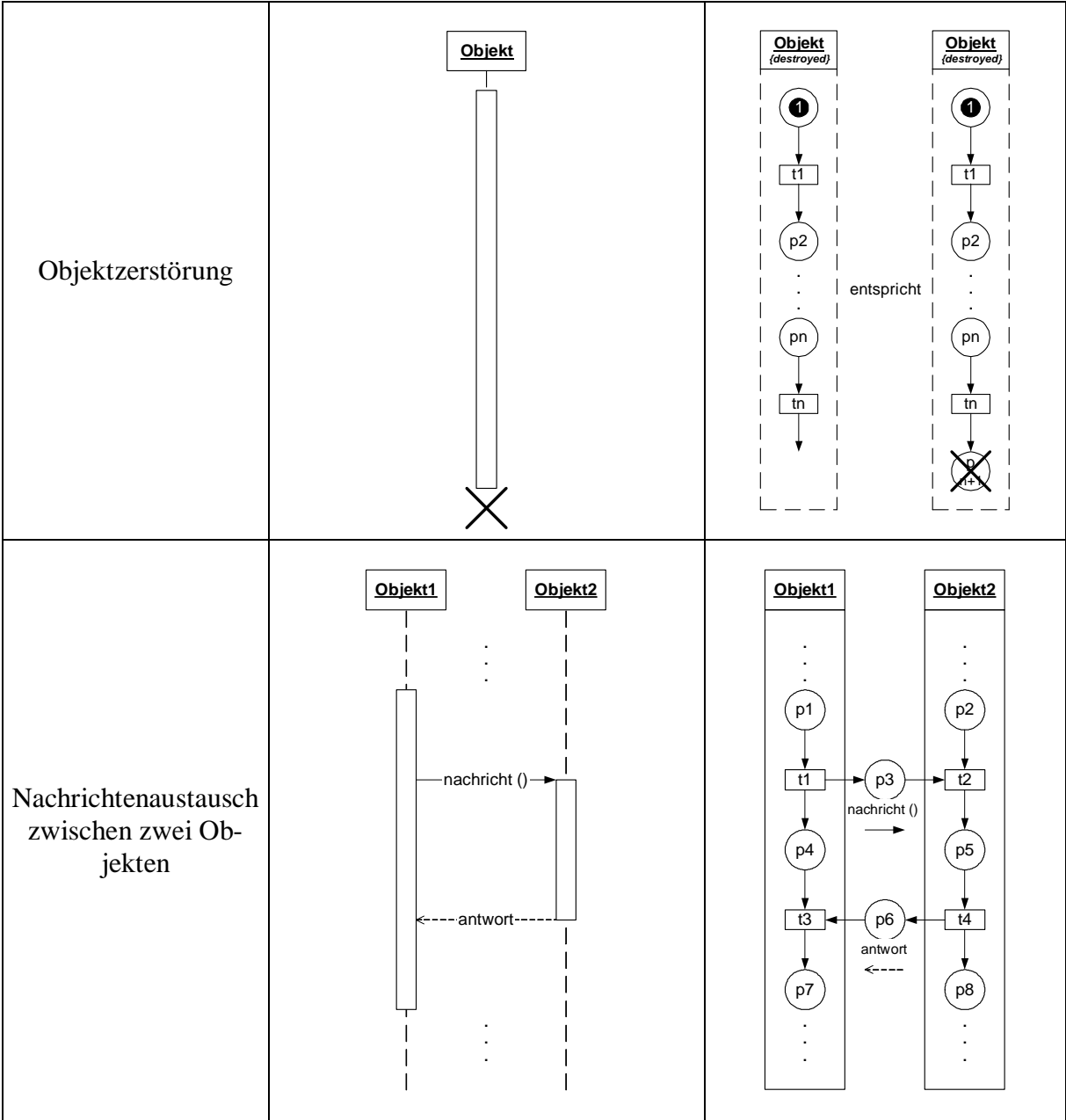
Elementsbezeichnung	Zustandsdiagramm-Baustein	Petri-Netz- Baustein
Zustand		
Einfache Transition		
Synchronisation (Gabelung)		
Synchronisation (Vereinigung)		

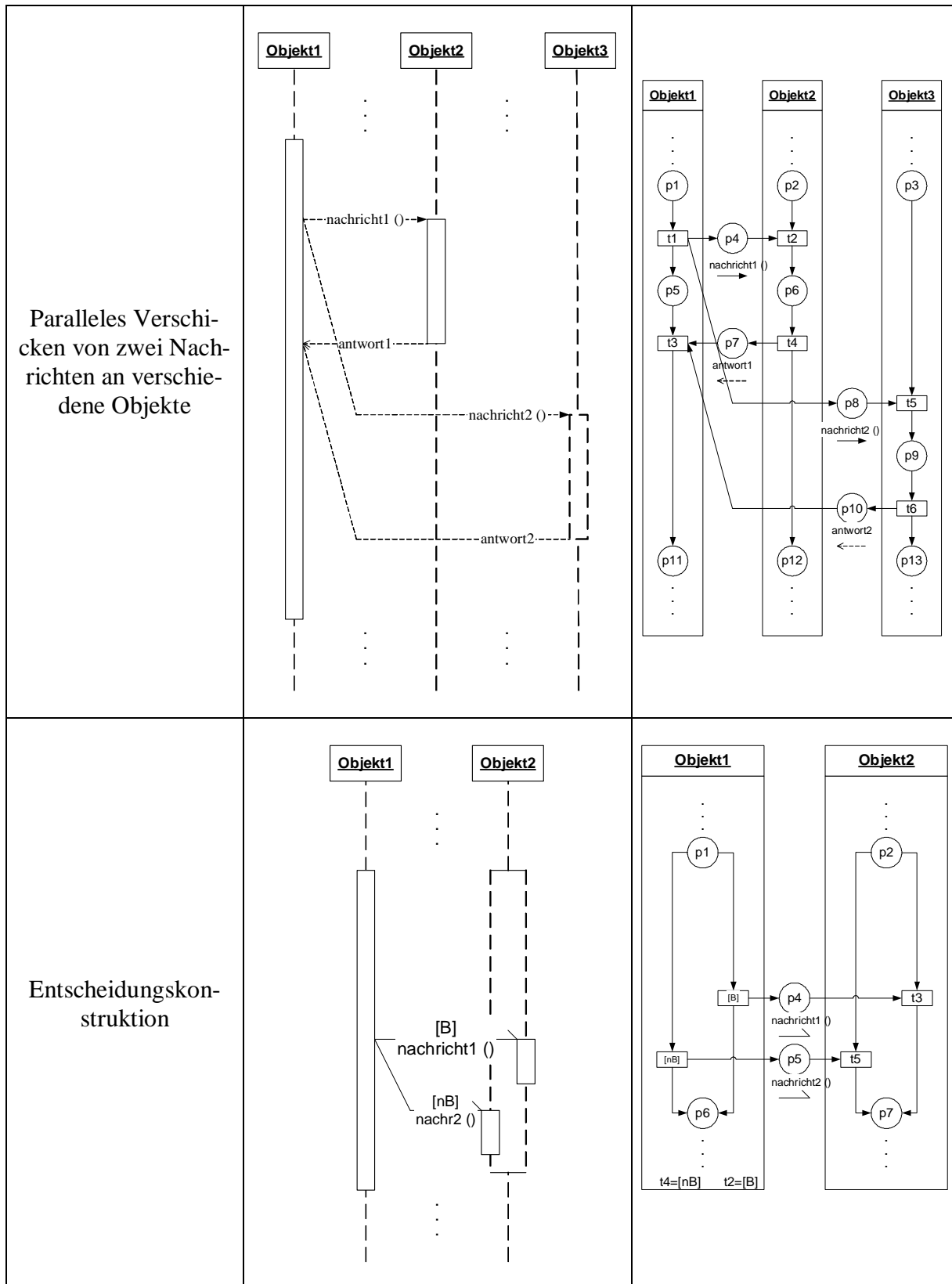
Die Transformationen von Message Sequence Charts in Petri-Netze sind schon in mehreren Arbeiten [GrRu 93, Lang 00, KIPaEh 00, Klug 00, KIHo 00, Klug 02, GeGo 99, YaTe 00, Stör 99] erarbeitet worden. In [Döri 02] findet man die Beschreibung von diesen Transformationsalgorithmen für Message Sequence Charts in Zeitintervall-Petri-Netze. In [Rang 2001] wird wieder die Vorgehensweise der Überführung von einzelnen Bausteinen der Sequenzdiagramme der UML in Bausteine von Petri-Netzen genutzt. Deshalb werden im Weiteren die in der nachfolgenden Tabelle 4 [Rang 2001] beschriebenen Überführungskonstruktionen als

Basis für die Transformationsalgorithmen von gefärbten Sequenzdiagrammen im Kapitel 5 in Gefärbte Petri-Netze benutzt.

Tabelle 4: Sequenzdiagramm- und Petri-Netz-Bausteine

Element	Sequenzdiagramm-Baustein	Petri-Netz-Baustein
Objekt		
Objekterzeugung		





Die Berücksichtigung von Zeiten [Döri 02] wird mit der Einführung eines zusätzlichen Platzes rechts neben der eigentlichen Instanz-Spalte realisiert. Die nachfolgende Transition wird dabei mit einem Zeitwert versehen, welcher einer rationalen Zahl bzw. einem Intervall rationaler Zahlen entspricht, welches eine zusätzliche Schaltbedingung an der Transition darstellt (t_v oder t_i).

Eine Rücktransformation von entstandenen Petri-Netzen in Diagramme der UML ist noch nicht vorgenommen worden.

2.6 Verifikationsmethoden

Verifikation wird zum Überprüfen der Korrektheit eines Entwurfs (z.B. von eingebetteten Echtzeitsystemen) und zum Nachweis von Fehlfunktionen eingesetzt. Dabei kann ein Modell des Systems durch Verifikation auf bestimmte Eigenschaften, z.B. Existenz von Nebenläufigkeiten oder das Einhalten von angegebenen Zeitrestriktionen, untersucht werden. Es wird auch überprüft, ob ein gewünschtes Systemverhalten eingehalten wird bzw. ob unerwünschtes, eventuell auch gefährliches Systemverhalten auftreten kann.

Der Begriff Verifikation hat verschiedene Definitionen:

”Verifikation ist die Überprüfung des Wahrheitsgehaltes eines Modells gegen seine Spezifikation.“

”Proof by formal or analytical means that the specification of a software system is consistent and complete.“ [VeTeReGr 2000]

Es gibt verschiedene Arten und Möglichkeiten zur Verifikation von eingebetteten Echtzeitsystemen, die sich in Komplexität und Aussagekraft unterscheiden. Dazu gehören Simulation, Test, deduktive Methoden (Theorem proofing) und Model checking oder formale Verifikation an Hand von formalen Modellierungsmitteln wie z.B. Petri-Netzen.

Unter Simulation versteht man „die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.“ [VDI-Ge]. Simulation dient dem Zweck, eine Prognose über das Verhalten des modellierten Systems zu erhalten. Um Simulation durchführen zu können, braucht man ein ausführbares Modell des Systems. Bei der Simulation können meistens nicht alle sondern nur bestimmte Fälle untersucht werden. Das führt dazu, dass im Allgemeinen nicht alle Fehler entdeckt werden können. Deshalb wird die Fehlerfreiheit des gesamten Systems dabei nicht garantiert.

Bei Testverfahren am realisierten System wird auch nur das Systemverhalten in ausgewählten Fällen untersucht. Es gilt damit auch die Einschränkung der Simulation der nicht garantierten Fehlerfreiheit.

Verifikationsverfahren wie Model checking [Clar 99], darunter auch Temporal model checking und Symbolic model checking, Ansatz von Automaten [CIEm 81, QuSi 82], unterstützt die Systemüberprüfung auf bestimmte Eigenschaften an Hand eines Modells. Da die Suche in einem Zustandsraum des Modells läuft, ergibt sie eine große Anzahl von Zuständen, wie sie in realen Systemen auftreten, obwohl es schon Tools gibt, welche Zustandsräume mit 10^{100} Elementen beherrschen. Diese Methode wird vorzugsweise bei der Verifikation von Hardware und Protokollen eingesetzt. Bei Temporal model checking wird die Erreichbarkeit einzelner Zustände im Gegensatz zum Symbolic model checking untersucht, wo das gesamte Systemverhalten berechnet wird. Als größere Anwendungsfälle für Model checking sind Überprüfungen des IEEE Futurebus+, IEEE SCI und ISDN [Clar 93; Dill 92; Chav 92] oder eines

Druckreglers eines elektronischen Bremssteuersystems für Nutzfahrzeuge [BiGu 00] durchgeführt worden.

Beim Theorem proofing [CIWi 96] wird die Spezifikation und das Modell mit Hilfe mathematischer Logik durch Formeln beschrieben. Das erlaubt die Analyse von Modellen mit unendlichem Zustandsraum. Die Verifikationsverfahren bestehen dabei aus dem Entwurf des Beweises für die verschiedenen Eigenschaften aus Axiomen. Systeme zur maschinellen Erstellung von Beweisen sind in [KaMo 95, Crai 88; Corn 95; Gord 87; LuPo 92; ClZh 93] beschrieben.

Als eine Menge weiterer Verifikationsmethoden ist die formale Verifikation mit Petri-Netz-Modellen verbreitet. Diese stellen mit mathematischen Beweisen sicher, dass ein System geforderte Eigenschaften erfüllt. Viele Methoden davon sind schon lange bekannt und ziemlich gut untersucht. Es werden aber ständig Weiterentwicklungen vorgenommen. Die Methoden wurden in größeren Projekten, wie von der Deutschen Telekom in einer Studie „Höhere Petrinetze im Bereich Intelligenter Netzwerke“ [Cape 99, Cape 98; CaDi 93], „WAP Class 2 Transaction Service“ [GoBi 00] und TCP-Protokollvarianten vom Hewlett-Packard CPN Centre [FiKr 99], angewendet.

Die Verifikation ohne Zeitbewertung liefert Aussagen zur Erreichbarkeit von Zuständen des modellierten Systems, zur Lebendigkeit, Konfliktfreiheit, Beschränktheit und Sicherheit. Die Verifikation des Zeitverhaltens eines Systems zeigt, ob die zeitlichen Restriktionen erfüllt werden. Dabei können sich aufgrund der Zeitbewertungen auch andere Aussagen zu den o.g. Eigenschaften ergeben [Popo 89, Popo 94, Popo 98, Döri 02, Lich 04]. Die in Quellen [Popo 97, Döri 02] beschriebene Methode wurde für Zeitintervall-Petri-Netze entwickelt und stellt eine Grundlage für die später in der Arbeit vorgenommene Erweiterung dieser Methode auf Gefärbte Zeitintervall-Petri-Netze dar. Um die Verifikation von Zeitaspekten des Systems zu ermöglichen, wurden so genannte Zustandsklassen für Zeitintervall-Petri-Netze definiert [Popo 89]:

$$C_0 := \left\{ z \mid \exists \tau \left(\tau \in Q_0 \wedge z_0 \xrightarrow{\tau} z \right) \right\}$$

$$C' := \left\{ z'' \mid \exists z \exists z' \exists \tau \left(z \in C \wedge \tau \in Q_0 \wedge z \xrightarrow{t} z' \xrightarrow{\tau} z'' \right) \right\}$$

wobei C_0 eine Zustandsklasse ist, welche den Startzustand enthält, und C' alle weiteren Zustandsklassen enthält. Dabei ist jede nachfolgende Zustandsklasse von Restriktionen der vorherigen Klasse abhängig. Deshalb sind in [Popo 99, Popo 89, Döri 02] zur Ableitung der Zustandsklassen aus der Ausgangsklasse (kanonische Parameterdarstellung genannt) durch die Angaben der Schaltsequenzen der schaltenden Transitionen und der Restriktionen der auftretenden Zeiten folgende Definitionen gegeben:

Sei $\left[\{u_t, v_t / t \in T\}, \{A^2\}, \{F^2\} \right]$ eine Signatur aus Konstanten, Prädikaten und Funktionen, wobei u_t - die Untergrenze für die Schaltfähigkeit und v_t - die Obergrenze für die Schaltfähigkeit sind, dann

ist $\Sigma = [Q_0, \omega]$ mit:

$$\begin{aligned}\omega(u_i) &:= \underline{I}(t), \\ \omega(v_i) &:= \bar{I}(t), \\ \omega(A^2) &:= " \leq ", \\ \omega(F^2) &:= " + " \end{aligned}$$

wobei $\underline{I}(t)$ - die Intervall-Startzeit und $\bar{I}(t)$ - die Intervall-Stoppzeit sind.

Weiter wird in diesen Arbeiten die partielle Funktion $\delta: T^2 \rightarrow R_{Z^N}(m_0) \times \mathbf{SUM}^T \times \mathbf{P}(\mathbf{BED})$, wobei \mathbf{SUM} eine Menge von Termen, die jede Variable höchstens einmal und keine Konstanten enthalten, und \mathbf{BED} eine Menge aller Terme in der Form $A^2 \text{term}_i \text{term}_j$, mit term_i aus \mathbf{SUM} und term_j eine Konstante oder umgekehrt, sind, definiert.

Dabei gilt für die erste partielle Abbildung $\delta(e) := \{m_0, \Sigma_0, B_0\}$:

1. $\Sigma_0(t) := \begin{cases} \tau_1, & \text{falls } t^- \leq m_0 \\ \#, & \text{sonst} \end{cases}$
2. $B_0 := \{A^2 \Sigma(t) v_i \mid t^- \leq m_0\}$

Bei jedem weiteren Schritt $\delta(w\hat{t}) := [m', \Sigma', B']$ mit $\delta(w) = [m, \Sigma, B]$ und $\hat{t} \in T, \hat{t}^- \leq m$ gilt:

1. $m' = m + \Delta \hat{t}$
2. $\Sigma'(t) := \begin{cases} \#, & \text{falls } t^- \not\leq m' \\ \tau_j, & \text{falls } (t^- \not\leq m \wedge t^- \leq m') \vee (t^- \leq m \wedge t^- \leq m'), \\ \Sigma(t) + \tau_j, & \text{sonst} \end{cases}$

wobei j ein kleinster Index der Variablen (j), welche weder in der Klammer noch in \mathbf{BED} auftritt und j größer als ein vorkommender Index ist.

3. $B' := B \cup \{A^2 u_i(\Sigma(\hat{t}))\} \cup \{A^2 \Sigma'(t) v_i \mid t^- \leq m'\}$

Mit diesem Algorithmus werden Zustandsklassenfolgen gebildet und die Zeiten für Transitionssequenzen festgestellt. Durch die Ermittlung von Grenzzuständen \underline{z} und \bar{z} der Zustandsklasse C , welche den Zuständen mit den kleinsten und größten Schaltzeiten aller Transitionen entsprechen und die kürzeste und längste Zeit für die ganze Transitionssequenz aufweisen, und der Untersuchung, ob sie im Erreichbarkeitsgraphen $\text{EG}_Z(z_0)$ erhalten sind, kann die Erreichbarkeit oder Nichterreichbarkeit von diesem Zustand bewiesen werden [Popo 89].

2.7 Schlussfolgerungen und Ziele

Die in dieser Arbeit zu entwickelnden Modellierungsmittel zum Entwurf von eingebetteten verteilten Echtzeitsystemen sollen auf die spezifischen Besonderheiten dieser Systeme optimiert werden. Diese sind vor allem Heterogenität und Komplexität, Nebenläufigkeit von Prozessen, sowohl parallele als auch sequentielle Ausführung, Verteilung und Echtzeitbedingungen. Als Grundlage wird die standardisierte, breit bekannte, gut strukturierte objektorientierte Sprachfamilie UML (Unified Modeling Language) zur grafischen Notation und Spezifikation eingesetzt. Die Möglichkeiten, die vielfältigen Aspekte eines Systems durch den Einsatz von verschiedenen Diagrammtypen darzustellen, haben dieser Sprache große Akzeptanz gebracht. Dennoch geht bei der Modellierung komplexer Systeme ziemlich schnell die Übersichtlich-

keit der dargestellten Systeme bzw. Prozesse in den Diagrammen verloren. Außerdem ist das grafische Konzept nicht durch einen mathematischen Formalismus unterstützt, was die formale Analyse der Systemeigenschaften verhindert. Es existiert auch kein Übergang von einem Beschreibungsmittel in ein anderes. Die neueste Version der UML unterstützt zwar ein Zeitkonzept, aber die Zeitangaben haben wegen fehlender formaler Semantik nur einen dokumentarischen Nutzen.

Da die drei dynamischen Diagrammtypen der UML, die Sequence Diagrams, die Statechart Diagrams und die Activity Diagrams im Modellierungsprozess für den Schwerpunkt der vorliegenden Arbeit, das dynamische Verhalten, genutzt werden, wird das vorgeschlagene Erweiterungs- und Transformationskonzept für diese Diagramme ausgearbeitet. Zur Realisierung dieses Konzeptes werden folgende Ziele in der Arbeit verfolgt:

- Ausarbeitung eines neuen Konzeptes zur gleichzeitigen Darstellung parallel laufender Prozesse in diesen Diagrammen durch Einführung von Farben, welche u. a. unterschiedliche Objekte bzw. Prozesse beschreiben,
- Formulierung der grundlegenden Definitionen der Syntaxerweiterung für die gefärbten Diagramme,
- Nachweis der Funktionalität,
- Transformation dieser Diagrammtypen in Gefärbte Petri-Netze, Definition von entstandenen eingeschränkten Petri-Netz-Klassen,
- Entwicklung von Transformationsregeln für Überführung dieser eingeschränkten Petri-Netze in andere Diagramme,
- Erweiterung der Verifikationsmethoden bezüglich Zeiteigenschaften für gefärbten Zustands-, Aktivitäts-, Sequenzmodelle.

Gefärbte und zeitbewertete Diagramme

Im Verlauf dieses Kapitels wird übersichtlich die Erweiterung der UML-Diagramme für die dynamische Modellierung, die Zustands-, Aktivitäts- und Sequenzdiagramme, um eine neue Eigenschaft – die Farbe vorgestellt. Diese erweiterten Diagrammtypen werden als gefärbte Diagramme bezeichnet. Dieses deutet auf die Übernahme von Prinzipien des etablierten Konzeptes der höheren Petri-Netze auf die auch außerhalb der UML weit verbreiteten Diagrammtypen hin. Die grundlegenden Definitionen und Syntaxerweiterungen für diese Diagrammtypen bilden einen zentralen Abschnitt dieser Arbeit. Durch die Zusammenfassung ähnlicher Abläufe in gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen wird gezeigt, wie parallele, teilweise identische Teilprozesse mit einer gemeinsamen Struktur modelliert werden können. Es wird die Zweckmäßigkeit der Einführung von Farben in den Diagrammen mit dem Ziel der Erhöhung des Informationsgehaltes bei Erhalt der Übersichtlichkeit des Modells nachgewiesen.

Zu Beginn erfolgen die wichtigsten Definitionen für die erweiterten Diagrammtypen. Dabei werden Analogien zum Farbkonzept von Petri-Netzen beschrieben. Das hat den Zweck, den Sinn der Farbe leichter zu verstehen. Weiterhin bereitet es die Übernahme der Farbeigenschaften bei der späteren Umwandlung in Gefärbte Petri-Netze vor. Die Faltungsmöglichkeiten für Objekte in gefärbten Diagrammen werden erläutert. Nach den grundlegenden Definitionen werden die einzelnen Modellierungselemente und mögliche Modellierungskonstrukte in erweiterter Form eingeführt. Danach werden die Erläuterungen zur Funktionsweise gegeben. Als wichtige Eigenschaft für Echtzeitsysteme wird das Zeitkonzept für die gefärbten Diagramme ausgearbeitet.

Um die Ähnlichkeit mit dem UML-Standard weitgehend zu erhalten, werden die Notationen und grafischen Darstellungen möglichst beibehalten, was die Verwendung der gefärbten Diagramme in existierenden Werkzeugen und Standards ermöglicht.

Im vorliegenden Kapitel werden die erweiterten Diagramme nur teilweise formal beschrieben, wie es auch in der UML für die ursprünglichen Diagramme erfolgt. Für die exakte Beschreibung des Verhaltens der einzelnen Elemente der gefärbten Diagramme und deren Zusammenwirken dient deshalb die in Kapitel 5 enthaltene Transformation in Gefärbte Petri-Netze, die dabei formal das Verhalten der Ausgangsdiagramme beschreiben.

3.1 Beschreibung von Varianten eines Modells in einem gefärbten Diagramm

Eine der Haupteigenschaften von eingebetteten verteilten Echtzeitsystemen ist die Nebenläufigkeit von Prozessen. Um diese Prozesse mittels standardisierten Zustands- oder Aktivitätsdiagrammen darzustellen, muss man mehrere Diagramme bilden: je eines pro Prozess und pro Kontrollfluss zwischen ihnen. Dabei haben diese Diagramme, die eigentlich gemeinsam betrachtet werden müssen, viele gleiche Elemente und Strukturen. Das ist visuell sehr unübersichtlich und aufwendig in der grafischen Darstellung. Um noch dazu die Abhängigkeiten und Verbindungen zwischen diesen Diagrammen zu zeigen, muss man den komplizierten Mechanismus des Ereignisaustausches durch Zwischenzustände benutzen. Deshalb ist auf diesem Gebiet ein Mechanismus für die leistungsfähige Modellierung sinnvoll, der mehrere Prozesse in einem Diagramm zu beobachten erlaubt.

Bei der Darstellung in Sequenzdiagrammen, die den Nachrichtenaustausch zwischen mehreren, ähnliche Funktionen erfüllenden, aber unterschiedlichen Objekten beschreiben, kommt man ziemlich schnell zu sehr großen und unübersichtlichen Diagrammen. Um das umgehen zu können wäre es hierbei vorteilhaft, die Referenzierung von Instanzen durchzuführen.

Der Ausweg aus dieser Problematik ist die Erweiterung der vorhandenen Standards auf gefärbte Diagramme. Diese entstehen durch *Faltung* von mehreren einfachen Diagrammen, die unabhängig ausgeführt werden, oder sich in einigen Teilen beeinflussen. Unter „Faltung“ wird dabei in Zustands-, Aktivitäts- und Sequenzdiagrammen das Überlagern von:

- Zuständen und Transitionen (Abbildung 28),
- Aktivitäten und Transitionen (Abbildung 29),
- Instanzen und Nachrichten (Abbildung 30)

ähnlichen Inhaltes und ähnlicher Struktur verstanden. Unter „ähnlich“ versteht man größtenteils einheitliches Verhalten mit wenigen Abweichungen.

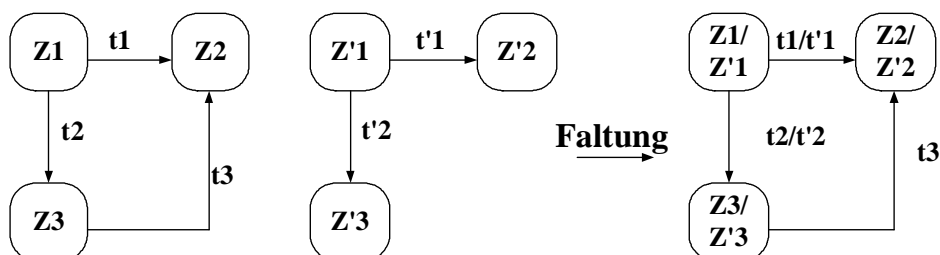


Abbildung 28: Beispiel zur Faltung von Zuständen und Transitionen

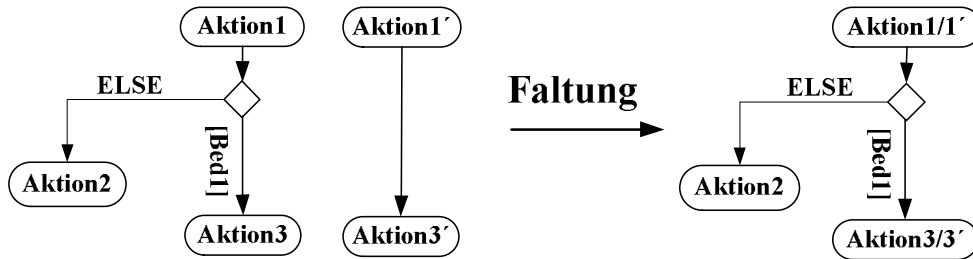


Abbildung 29: Beispiel zu Faltung von Aktivitäten und Transitionen

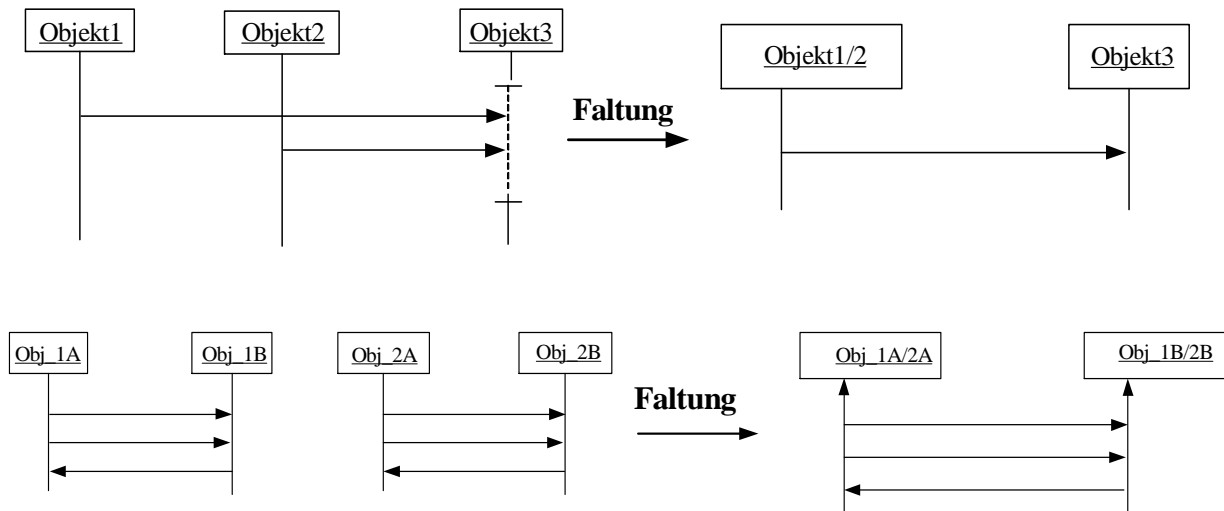


Abbildung 30: Beispiel zu Faltungsmöglichkeiten in Sequenzdiagrammen

Da diese Diagramme in der objektorientierten Entwicklung eingesetzt werden, kann der Ähnlichkeitsbegriff auf das Klassenkonzept übertragen werden. Dabei werden sie in der UML zur Beschreibung des Verhaltens einer Klasse eingesetzt. Beschreiben zwei Diagramme zwei verschiedene Unterklassen, lassen sich diese beiden z.B. zu einem gefärbten Diagramm zusammenfassen. Es ergeben sich hierbei folgende Faltungsmöglichkeiten:

- mehrere Unterklassen,
- eine Klasse und deren Unterklassen,
- mehrere Objekte einer Klasse,
- mehrere Objekte verschiedener Unterklassen (mit gleicher Elternklasse).

Die gefärbten Zustands-, Aktivitäts- und Sequenzdiagramme widmen sich damit der Darstellung von komplexen dynamischen Prozessabläufen bei Beteiligung mehrerer Objekte (Instanzen) mit ähnlichem Verhalten und zusammenhängenden Beziehungen zwischen ihnen.

Die Objekte stellen dabei aktive Modellelemente dar, welche als Exemplare bestimmter Klassen gekennzeichnet sind und Attribute dieser Klasse besitzen. Die Objekte werden in gefärbten Diagrammen in Anlehnung an die Theorie höherer Petri-Netze durch gefärbte Marken oder einfach durch die Farben unterschieden. Die Farbe modelliert einen Typ, d. h. eine symbolische Bezeichnung des Objektes (Prozesses). Deshalb sind Farben von Marken nicht unbedingt natürliche Farben. Eine konkrete Marke einer Farbe beschreibt den aktuellen lokalen Zustand des entsprechenden Objektes (Prozesses). Beispielweise bedeutet für die Aktivitäts-

diagramme der Ausdruck „Die Marke mit roter Farbe ist in Aktion A“, dass die Aktion A für den Prozess, der als „rot“ bezeichnet wird, ausgeführt wird. Für die Zustandsdiagramme bedeutet die Belegung des Zustandes mit verschiedenen Marken, dass dieser Zustand für verschiedene Objekte gerade aktiv ist. Ist ein Objektsymbol in Sequenzdiagrammen mit mehreren Farben versehen, bedeutet dieses, dass n verschiedenfarbige, aber ähnliche Objekte sich gleichzeitig in der Lebenslinie befinden können und am Verschicken von Nachrichten teilnehmen können. Dabei ist der Fluss der Marke einer Farbe in diesen Diagrammen eine visuelle Darstellung des Kontrollflusses für das entsprechende Objekt (den Prozess). Die Darstellung von diesen gefärbten Marken oder Farben ist in zwei Varianten vorgesehen (Abbildung 31): als eine grafische oder eine textuelle Notation, welche abhängig von der Anzahl der Farben oder anderen pragmatischen Bedingungen benutzt werden können.

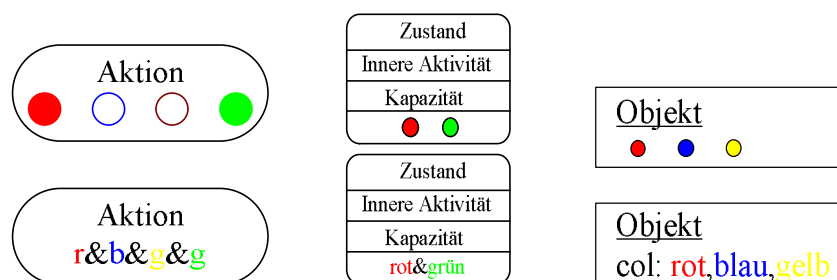


Abbildung 31: Notationsvarianten für gefärbte Belegungen von Aktionen, Zuständen und Instanzen



Abbildung 32: Weitere Notationsvariante für die Darstellung von gefärbten Aktionen, Zuständen und Instanzen

In Abbildung 32 (nach [Riab 2001], [Slav 2001], [Rang2001]) ist eine sehr übersichtliche Notationsweise für gefärbte Aktionen, Zustände und Instanzen dargestellt, welche die Zusammenfassung mehrerer zu einer multiplen Aktion, einem Zustand oder einer Instanz zeigt. Diese wird für die Anwendung in neuen Tools empfohlen. Da diese Variante nicht von allen bisher vorhandenen Tools unterstützt werden kann, wird im weiteren die Notationsweise aus Abbildung 31 übernommen.

3.2 Gefärbte Zustandsdiagramme

Die Erweiterung eines Zustandsdiagrammes um Farben ermöglicht die gleichzeitige Ausführung von mehreren Aktivitäten, die Einführung von komplexen Zustandskonfigurationen, in welchen mehrere lokale Zustände gleichzeitig aktiv sein können, und das Schalten von gefärbten Transitionen, das dem gleichzeitigen Schalten von mehreren einfachen Transitionen bzw. dem mehrfachen Schalten einer Transition in verschiedenen Farben entspricht.

3.2.1 Definition

Def. Ein gefärbtes Zustandsdiagramm beschreibt die möglichen Folgen von Zuständen, welche mehrere Objekte mit ähnlichem Verhalten, die zu einer Klasse oder verschiedenen Unterklassen einer Klasse gehören, der Reihe nach einnehmen können (Abbildung 33). Zur Darstellung des dynamischen Verhaltens mehrerer Objekte besteht ein gefärbtes Zustandsdiagramm aus:

- Einem komplexen Anfangszustand, welcher für mehrere Objekte gelten kann;
- einer endlichen Menge von Objekten, die durch die Farben (Typen) unterschieden werden,
- einer endlichen Menge von komplexen Zuständen, welche mehrere lokale Zustände von Objekten zusammenfassen;
- einer endlichen Menge von komplexen Ereignissen, welche für einen oder mehrere Objekte eintreten können;
- einer endlichen Anzahl von Transitionen, die die Übergänge zwischen Zuständen darstellen und durch zusätzliche Bedingungen bezogen auf ausgewählte Objekte zum Übergang aus einem Zustand in einen anderen gekennzeichnet sein können,
- einem oder mehreren Endzuständen.

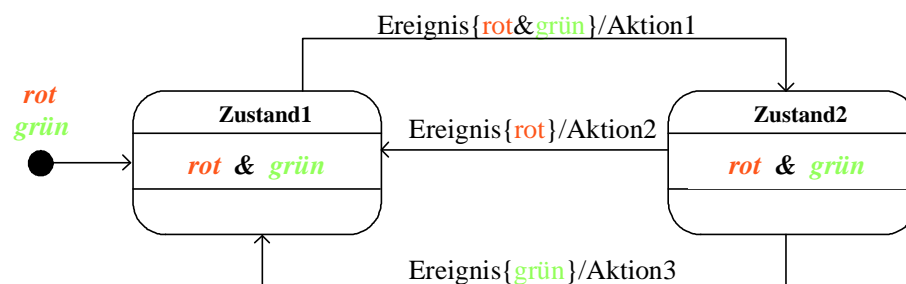


Abbildung 33: Beispiel für ein gefärbtes Zustandsdiagramm

Das gefärbte Zustandsdiagramm wird durch einen gerichteten Graph mit Knoten als Systemzustände und Kanten als Transitionen zwischen den Zuständen repräsentiert. Dabei besteht die Erweiterung der Zustandsdiagramme der UML auf gefärbte Zustandsdiagramme in dem möglichen Vorhandensein mehrerer Objekte in einem Zustand, welche durch verschiedene Farben unterschieden werden und durch farbige Marken verkörpert sind. Dazu kommen noch zusätzliche farbabhängige Transitionsbedingungen an Transitionen.

3.2.2 Syntaxerweiterung

Ein Zustandsdiagramm besteht hauptsächlich aus Zuständen und Zustandsübergängen. So werden die Erweiterungen für diese Elemente gesondert behandelt und anschließend erklärt, wie sich das gesamte Verhalten des Zustandsdiagrammes dabei ändert.

3.2.2.1 Der gefärbte Zustand

Wie schon oben erwähnt wurde, stellt ein gefärbter Zustand eine Faltung von mehreren lokalen, im UML-Standard als einfache normale Zustände betrachteten Zuständen dar. Der für verschiedene Objekte aktuelle Zustand wird durch Einführung zusätzlicher Felder in der Notation dargestellt. Diese zwei zusätzlichen Felder sind für die Kennzeichnung der gesamten Kapazität des Zustandes und für die aktuelle Belegung dieses Zustands geschaffen worden. Die Kapazitätsfunktion wird aus der logischen Verknüpfung von verschiedenen Farben gebildet. Dabei kann jede Farbe im Gegensatz zur Theorie von Petri-Netzen nur einfach auftreten, da sie die genauere Abbildung von einem bestimmten Objekt ist. Die Kapazität zeigt mögliche Farbzusammensetzungen für einen Zustand, z.B.:

rot UND (*schwarz* ODER *blau*).

Das entspricht der gleichzeitigen Belegung dieses Zustands für die Objekte, welche durch rot und schwarz oder durch rot und blau gekennzeichnet sind. Ein weiteres zusätzliches Feld ist für die aktuelle Belegung des Zustandes reserviert. Die Erweiterung um diese zwei zusätzlichen rechteckigen Felder hat jetzt folgende Darstellung (Abbildung 34 (nach [Slav 2001])):

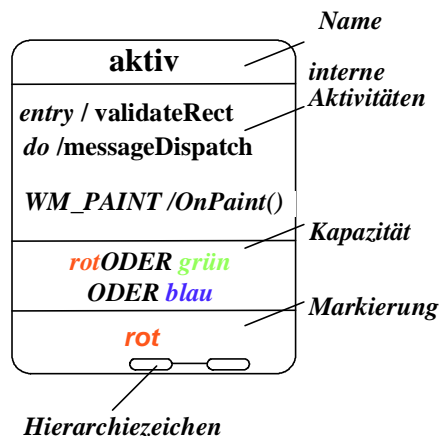


Abbildung 34: Beispiel zu einem gefärbten Zustand

Die drei Farben rot, grün oder blau stellen drei Objekte dar, welche der Zustand AKTIV nur exklusiv annehmen kann. Das Markierungsfeld zeigt dabei, für welches Objekt dieser Zustand gerade aktiv ist. Die vordefinierten Aktionen dieses Zustands sind für alle drei Objekte gleich.

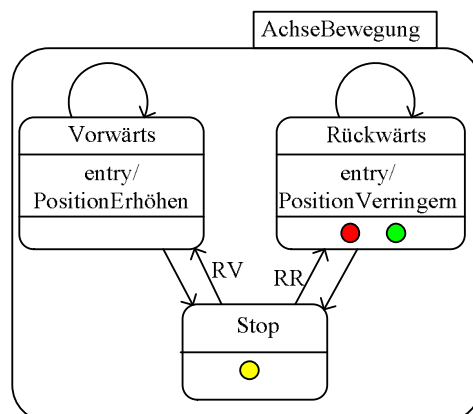


Abbildung 35: Ein gefärbtes Zustandsdiagramm aus dem Beispiel „Messsystem“

In der Abbildung 35 ist ein gefärbtes Diagramm für das Modell von der Bewegung von drei Achsen (Achse X, Achse Y1 und Achse Y2, die entsprechend mit gelber, roter und grüner Marke dargestellt sind) des in Kapitel 2.3.1 beschriebenen Messsystems, welches drei komplexe Zustände Vorwärts, Rückwärts und Stop enthält, gezeigt. Im Vergleich mit dem UML-Standard müsste man dieses Bewegungsverhalten von drei Achsen dort mit drei unterschiedlichen Zustandsdiagrammen modellieren.

Die Pseudozustände Start und History stellen in gefärbten Diagrammen auch farbabhängige Zustände dar. Die symbolische Darstellung des Startzustands bleibt erhalten: er wird durch einen schwarz ausgefüllten Kreis gekennzeichnet. Es kommt aber noch die Anschrift mit den Objekten, welche sich in diesem Startzustand im laufenden System befinden, dazu (Abbildung 36).

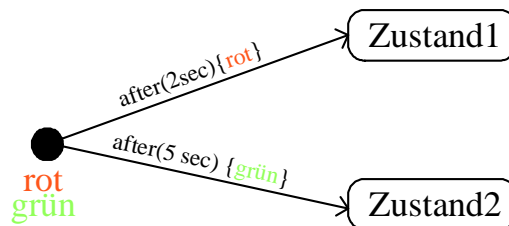


Abbildung 36: Beispiel zu einem gefärbten Startzustand

Das Vorhandensein von mehreren Objekten in einem Startzustand, welcher eigentlich mehrere einzelne Startzustände für bestimmte Farben zusammenfasst, kann zu mehreren Nachtransitionen führen. Die dem Initialzustand folgenden Transitionen können mit Angaben von Ereignissen versehen sein, dürfen aber keine Bedingungen oder Aktionen besitzen.

History-Zustände stellen ein Gedächtniskonzept für eine ODER-Verfeinerung dar. Sie beinhalten Informationen über die zuletzt ausgeführten Unterzustände, und bei erneutem Eintreten in diese Verfeinerung über den History-Zustand können aus diesem History-Zustand mehrere Transitionen, welche für verschiedene, sich gegenseitig ausschließende Objekte markiert sein können, ausgehen.

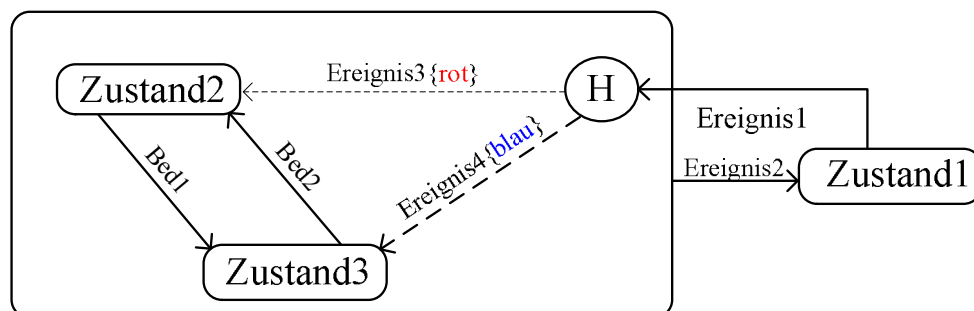


Abbildung 37: Beispiel für einen gefärbten History-Zustand

Bei sequentiellen (ODER) und parallelen (UND) Verfeinerungen müssen die Kapazitäten von Unterzuständen auf alle Farben, welche im Oberzustand vorkommen, angepasst werden (Abbildungen 37, 38, 39).

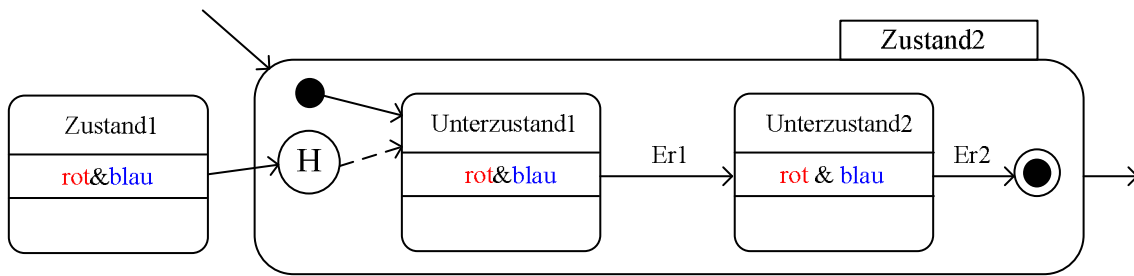


Abbildung 38: Beispiel zur ODER-Verfeinerung in gefärbten Zustandsdiagrammen

Dabei muss die Notation von Unterzuständen mit der Notation von Zuständen übereinstimmen. Die Transitionen, die zu Unterzuständen führen, dürfen nur von folgenden Zuständen ausgehen:

- aus einem beliebigen Unterzustand desselben Oberzustandes,
- aus dem Startzustand desselben zusammengesetzten Zustandes,
- aus dem History-Pseudozustand desselben Oberzustandes.

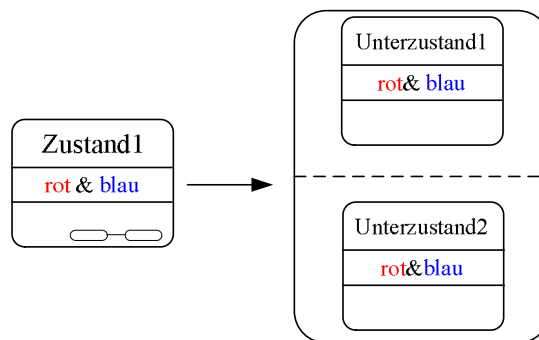


Abbildung 39: UND-Verfeinerung für Zustand1

3.2.2.2 Die gefärbte Transition

Die Transitionen, wie sie in Kapitel 2 schon beschrieben wurden, stellen die durch Ereignisse gesteuerten Zustandsübergänge dar. Das Ereigniskonzept bleibt bei gefärbten Diagrammen erhalten. Als Erweiterung erhält hier aber ein Ereignis außer den Parametern noch Verweise auf konkrete Objekte, für welche dieses Ereignis gilt. Das Format des Ereignisses wird folgendermaßen festgelegt:

event-name (' *comma-separated-parameter-list* ') { *Empfänger* }, wobei

event-name ein Ereignisname,

comma-separated-parameter-list eine Liste von Parametern,

Empfänger – ein oder mehrere Objekte, welche durch dieses Ereignis angesprochen werden, sind.

Der Parameter wird aus folgenden Teilen zusammengesetzt:

parameter-name ':' *type-expression*, wobei *parameter-name* ein Parametername und *type-expression* ein Ereignistyp ist.

In gefärbten Diagrammen werden von der Farbe abhängige Transitionen eingeführt. Dabei entspricht das Schalten einer solchen Transition dem gleichzeitigen Schalten von mehreren einfachen Transitionen bzw. dem mehrfachen Schalten einer Transition in verschiedenen Far-

ben. Die Überwachungsbedingungen von Transitionen werden mit der Möglichkeit eines Verweises auf bestimmte Objekte versehen (Abbildung 40). Entsprechend der UML werden Transitionen durch nach UML-Format beschriftete durchgezogene Pfeile, welche einen Zustand mit dessen Folgezustand verbinden, dargestellt,



Abbildung 40: Beispiel für eine erweiterte Transitionsspezifikation

Die Überwachungsbedingung der Transition kann bei gefärbten Zustandsdiagrammen auch einen zusätzlichen Verweis auf Objekte, die diese Transition durchlaufen dürfen, besitzen. Die verschiedenen Modellierungsfälle werden mit Hilfe von booleschen Funktionen, welche genaue Bedingungen für verschiedene Objekte bei Zustandsänderung aufstellen, angegeben.

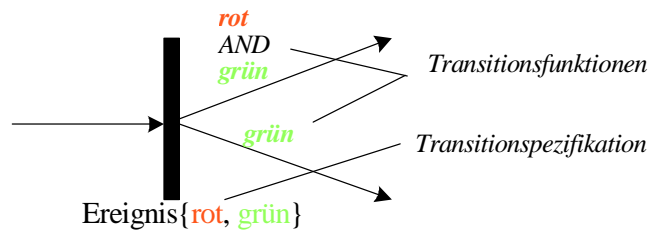


Abbildung 41: Beispiel für eine komplexe Transition, welche eine Verzweigung darstellt

Die komplexen Transitionen werden durch Synchronisationsbalken, welche mit der Transitionsspezifikation versehen sind und zur Modellierung von Synchronisationen oder Verzweigungen von Zuständen dienen, realisiert. Die ausgehenden Pfeile werden mit der Angabe von Objekten, welche sie passieren dürfen, dargestellt (Abbildungen 41, 42, 43).

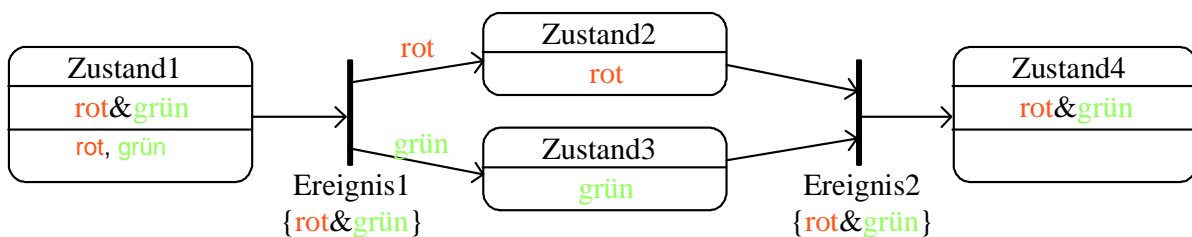


Abbildung 42: Darstellungsbeispiel von Synchronisationselementen

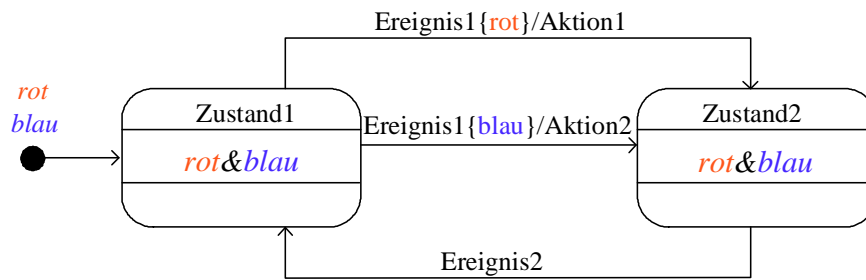


Abbildung 43: Verschicken desselben Ereignisses an verschiedene Objekte

Da in gefärbten Diagrammen ein Ereignis für verschiedene Objekte gültig sein kann, kann es deshalb an mehreren Transitionen mit der Voraussetzung vorkommen, dass es für unterschiedliche Objekte gesandt wurde.

3.2.3 Zeitbewertung

In den Zustandsdiagrammen der UML sind so genannte Zeitereignisse, welche das Erreichen eines bestimmten, durch Zeitausdruck spezifizierten Zeitpunktes modellieren, definiert (Abbildung 44 nach [Slav 2001]). Dabei stimmen die Zeitpunkte des Eintretens des Ereignisses und des Empfangs des Ereignisses überein. Die ausgehenden Transitionen werden durch die Zeitereignisse gesteuert, wenn in einem Modell definiert ist, dass sich ein Objekt in einem Zustand für eine bestimmte Zeit befinden soll oder wenn nach dieser Zeit eine zyklische Aktion durchgeführt werden soll. Solche Zeitereignisse werden öfters als Signale zur zeitlichen Steuerung von Prozessen bezeichnet.

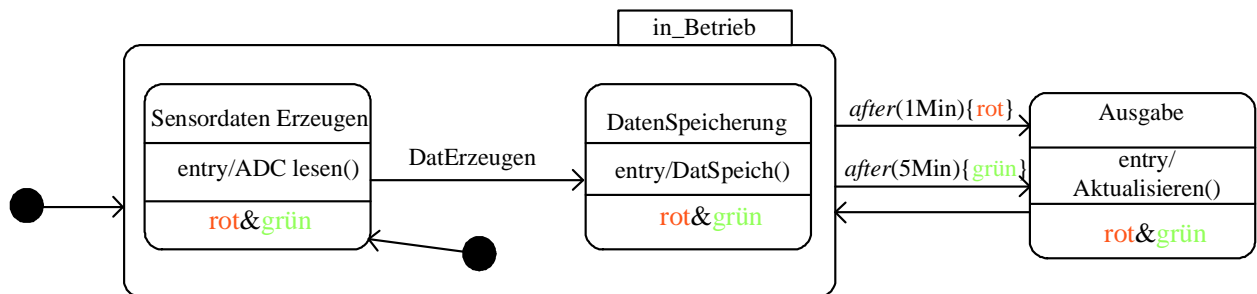


Abbildung 44: Zeitereignisse im gefärbten Zustandsdiagramm

Das in Abbildung 44 dargestellte gefärbte Zustandsdiagramm beschreibt das Verhalten einer einfachen Wetterstation. Dabei ist die Farbe **rot** die Luftdruckanzeige und die Farbe **grün** die Temperaturanzeige, welche in unterschiedlichen Zeitintervallen aktualisiert werden. *After* ist ein vorgegebenes Schlüsselwort zur Bezeichnung von Zeitereignissen, welches als Argument eine bestimmte Zeitspanne als Konstante enthält.

In dem UML-Standard ist eine optionale Zeit für einen Zustandsübergang, z.B. im Fall einer in der Transitionspezifikation angegebenen Aktion, welche bei der Ausführung eine bestimmte Zeit benötigt, vorgesehen. Wenn keine Zeit in der Transitionspezifikation angegeben ist, erfolgen Zustandsänderungen gewöhnlich „augenblicklich“. Der Zeitpunkt, in welchem ein Zustandsübergang und damit verbundenen das Senden von Nachrichten geschieht oder die bei

der Ausführung des Zustandsübergangs benötigte Zeit können aber genau angegeben werden. Zwar enthält die UML-Dokumentation keine genaue Angabe zur Position der Transitionszeit innerhalb der Transitionsspezifikation, aber die Möglichkeit einer solchen Spezifikation ist vordefiniert. Man kann die Transitionszeit am Ende der normalen Transitionsspezifikation nach dem Schrägstrich mit dem Schlüsselwort *Transitionszeit* angeben. In den gefärbten Diagrammen ist die Transitionszeit dabei farbabhängig. Sie kann für eine oder mehrere Farben gelten. Im folgenden Beispiel gilt sie nur für grün:

Ereignis(**rot**&**grün**) [Bedingung] / Aktion(**grün**) \ *Transitionszeit*{**grün**}[5,10]sec.

Es ist möglich, für unterschiedliche Farben bzw. Farbgruppen unterschiedliche Transitionszeiten zu definieren, die dann nur für die den Farben zugeordneten Objekte gelten.

3.2.4 Funktionsweise

Zu wichtigen Merkmalen von gefärbten Transitionen gehören folgende:

- Die Transitionen können durch komplexe Ereignisse gesteuert werden.
- Die Angabe von komplexen Bedingungen ist vorgesehen.
- Es können farbabhängig verschiedene Schaltmodi realisiert werden.
- Die Modellierung unterschiedlicher Schaltzeiten für verschiedene Objekte wird unterstützt.

Die Funktionsweise von gefärbten Zustandsdiagrammen wird hier an Hand eines Beispiels gezeigt. In diesem Beispiel kommen verschiedene Zustände vor, unter welchen sich auch Unterzustände und ein gefärbter Synchronisationszustand befinden. Dabei sind an diesem Diagrammausschnitt nur zwei Objekte, welche durch die Farben rot und grün unterschieden werden, beteiligt (Abbildung 45 nach [Slav 2001]).

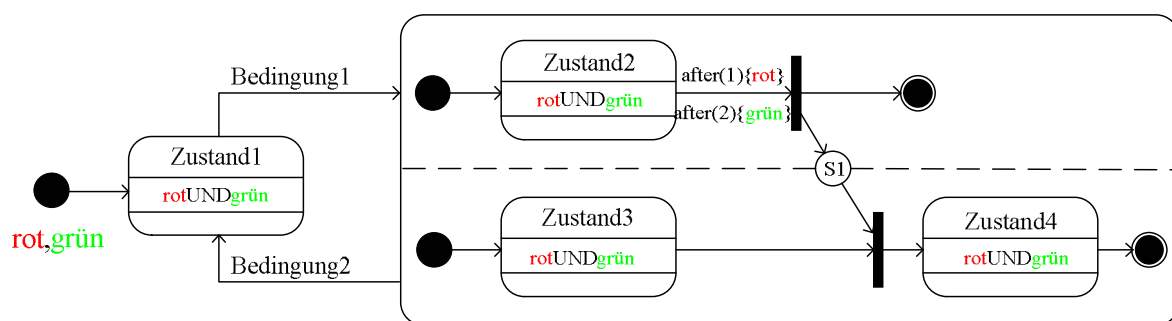


Abbildung 45: Beispiel für ein gefärbtes Zustandsdiagramm

Zustand1 kann mit beiden oder mit einer von beiden Farben belegt sein, dabei gelten die Bedingungen 1 und 2 gleichermaßen für beide Objekte, welche durch die Farben rot und grün dargestellt sind. Die Unterzustände einer parallelen UND-Verfeinerung sind auch für beide Farben relevant. Das Verlassen von Zustand2 wird durch farbabhängige Zeitereignisse gesteuert. S1 stellt einen gefärbten Synchronisationszustand dar. Dieser Synchronisationszustand wird von roten oder grünen Marken, die in Zustand2 vorkommen, beeinflusst.

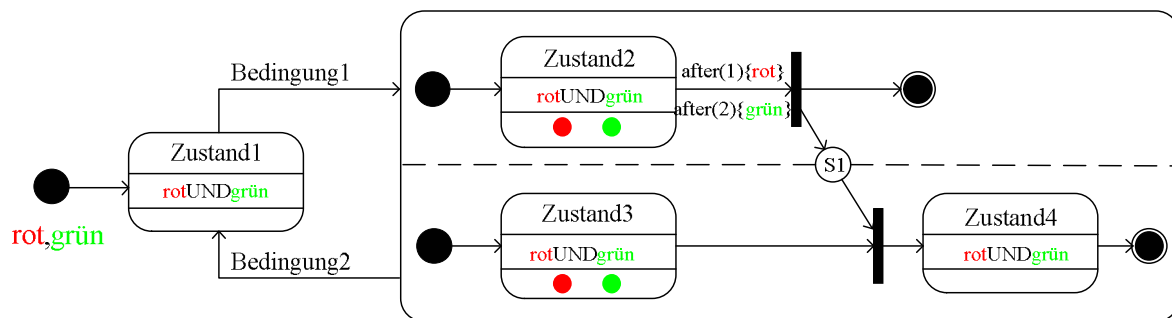


Abbildung 46: Ein weiteres Beispiel für ein gefärbtes Zustandsdiagramm

Aus der in Abbildung 46 gezeigten Situation werden folgende Zustandsübergangsfolgen möglich: nach einer Zeiteinheit verläßt die rote Marke Zustand2 und setzt den Synchronisationszustand S1 aktiv was eine Voraussetzung zum Schalten des Zustandsübergangs zwischen den Zuständen 3 und 4 für die rote Marke darstellt. Nach Ablauf von zwei Zeiteinheiten für die grüne Marke kann dieser Übergang weiter für die grüne Marke schalten. Danach befinden sich beide Marken in Zustand 4.

3.3 Gefärbte Aktivitätsdiagramme

Die gefärbten Aktivitätsdiagramme stellen eine Faltung von mehreren einfachen Diagrammen, die unabhängig ausgeführt werden, oder sich in einigen Teilen beeinflussen, dar. Dabei hat das gefärbte Aktivitätsdiagramm mehrere Kontrollflüsse, deren Anzahl durch die in diesem Diagramm beschriebenen Prozesse bestimmt wird. Jeder im Aktivitätsdiagramm dargestellte Prozess wird abstrakt durch eine ihm zugeordnete Farbe repräsentiert.

Hauptsächlich besteht ein Aktivitätsdiagramm aus Aktionen und Transitionen. Da die Transitionen in Aktivitätsdiagrammen viele verschiedene Entscheidungs-, Synchronisations- und Verzweigungselemente aufweisen können, werden diese Elemente gesondert weiter dargestellt.

3.3.1 Definitionen zum Diagramm

Def. Ein gefärbtes Aktivitätsdiagramm beschreibt die Ablaufmöglichkeiten verschiedener Varianten eines Systems mit Hilfe von farbabhängigen Aktivitäten, welche mehrere Prozesse mit ähnlichem Verhalten zusammenfassen, und mehreren farbabhängigen Kontrollflüssen. Ein gefärbtes Aktivitätsdiagramm besteht aus:

- Einem komplexen Anfangszustand, welcher für mehrere Objekte gelten kann;
- einer endlichen Menge von Prozessen, die durch die Farben (Typen) unterschieden werden,
- einer endlichen Menge von komplexen Aktivitäten, welche mehrere lokale Aktivitäten von verschiedenen Prozessen zusammenfassen;

- einer endlichen Menge von komplexen Transitionen, die Übergänge zwischen Aktivitäten darstellen und die durch zusätzliche Bedingungen, bezogen auf bestimmte Prozesse, zum Übergang aus einer Aktivität in eine andere gesteuert werden können,
- einem komplexen Endzustand, welcher den Endzustand für mehrere Farben darstellen kann .

Das gefärbte Aktivitätsdiagramm wird durch einen gerichteten Graph mit Knoten als Aktivitäten und Kanten als Transitionen zwischen Aktivitäten repräsentiert. Dabei besteht die Erweiterung gegenüber den Aktivitätsdiagrammen der UML zu gefärbten Aktivitätsdiagrammen in dem Modellieren von mehreren Prozessen mit verschiedenen Ablaufmöglichkeiten in einem Diagramm. Diese werden durch verschiedene Farben unterschieden und durch farbige Marken verkörpert. Die Anwendung von gefärbten Aktivitätsdiagrammen ist besonders vorteilhaft bei ähnlich verlaufenden Prozessen und bei dem Vorhandensein von vielen voneinander abhängigen Aktivitäten. Das erweitert dabei den Anwendungsbereich von Aktivitätsdiagrammen, bietet neue Modellierungsmöglichkeiten und verbessert die Übersichtlichkeit der Modelle. Außerdem helfen die gefärbten Aktivitätsdiagramme, redundante Bestandteile der Grafik zu vermeiden.

In Abbildung 47 ist ein kleiner Ausschnitt aus einem etwas komplexeren gefärbten Aktivitätsdiagramm, einem Beispielsmodell einer Bibliothek, dargestellt. Hier sind mehrere Elemente von gefärbten Aktivitätsdiagrammen, wie komplexe Aktionen mit bestimmten Aktionskapazitäten, farbabhängige Transitionen und komplexere Start- und Endzustände gezeigt.

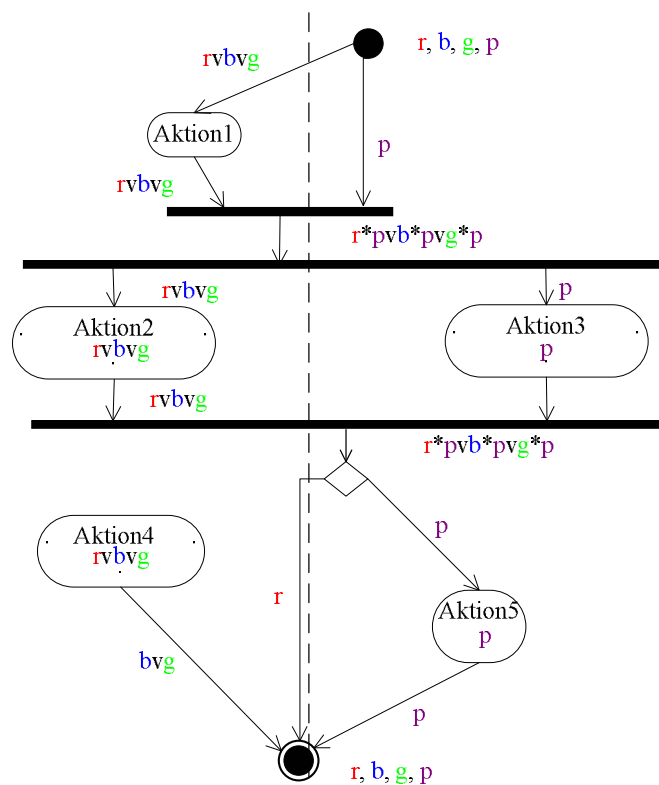


Abbildung 47: Ausschnitt aus einem komplexeren gefärbten Aktivitätsdiagramm

3.3.5 Syntaxerweiterung

Es werden hier die Erweiterungen für die Elemente des Aktivitätsdiagramms und Änderungen in gesamten Verhalten erläutert.

3.3.5.1 Die gefärbte Aktion

Die gefärbte Aktion stellt eine Faltung von mehreren einfachen Aktionen dar. Sie kann gleichzeitig für verschiedene Farben ausgeführt werden. Für die Modellierung von komplexem Verhalten von Systemen wird die Aktionskapazität, welche mit Hilfe von booleschen Ausdrücken das Vorhandensein von verschiedenen Farbkombinationen in einem Aktionszustand bestimmt, eingeführt. Z.B.:

$g\&b\&r$ $g\vee b\vee r$ $g\&(b\vee r)$

Der erste Kapazitätsausdruck verweist auf die Möglichkeit der gleichzeitigen Ausführung der Aktion für die Farben grün, blau und rot. Der zweite Kapazitätsausdruck definiert, dass die Farben nur exklusiv in dieser Aktion erscheinen können, was bedeutet, dass die gleichzeitige Ausführung dieser Aktion nur für eine Farbe möglich ist. Der dritte Kapazitätsausdruck zeigt, dass die Aktion für zwei verschiedene Farben: grün und blau oder für grün und rot gleichzeitig ausgeführt werden kann.

In der Notationsweise erscheint in der gefärbten Aktion noch zusätzlich ein Aktionskapazitätsausdruck, welcher grafisch durch gefärbte Kreise und Kreislinien innerhalb des Symbols oder symbolisch durch Angabe eines Kapazitätsausdrucks dargestellt werden kann (Abbildung 48 [Riab 2001]):

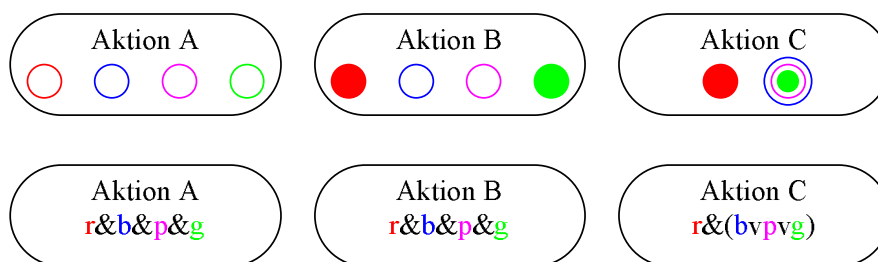


Abbildung 48: Angabemöglichkeiten von Kapazitäten in gefärbten Aktionen

In diesen Beispielen kann Aktion A gleichzeitig für rote, blaue, pink und grüne Marken ausgeführt werden. Die nicht ausgefüllten Kreise zeigen, dass keine Marke dieser Farbe zur Zeit in dieser Aktion vorhanden ist. Aktion B kann gleichzeitig für rote, blaue, pink und grüne Farben ausgeführt werden, dabei sind die Prozesse, welche durch rote und grüne Farben gekennzeichnet sind, in dieser Aktion aktiv. Aktion C kann für zwei Farben gleichzeitig ausgeführt sein: für rote und eine von drei Farben, die in der Klammer angegeben sind.

3.3.5.2 Die gefärbte Transition

Die Transition im gefärbten Aktivitätsdiagramm ist eine Faltung von mehreren einfachen Transitionen, von denen jede nur für ihre definierte Farbe schalten kann. Die vorgegebene Transitionsfunktion bestimmt, ob die Transition für jede Farbe unabhängig von anderen Far-

ben schalten kann oder für bestimmte Farben gleichzeitig schalten muss. Im Übrigen verhalten sich die Transitionen in gefärbten Aktivitätsdiagrammen wie Transitionen in normalen UML-Aktivitätsdiagrammen.

Eine Transitionsfunktion ist ein boolescher Ausdruck, der mit der Transition verbunden ist und die Schaltregeln definiert. Dabei treten als Argumente der Transitionsfunktion die Farben auf. Wird eine Aktion für eine bestimmte Farbe im Quellzustand beendet, so wird der Wert des entsprechenden Argumentes wahr. Wird in diesem Moment der Wert der gesamten Transitionsfunktion wahr, dann erfolgt der Aktivitätsübergang für alle Farben, die schon bearbeitet sind (Abbildung 49).



Abbildung 49: Notationsbeispiele für eine Transitionsfunktion

Die gefärbten Aktivitätsdiagramme bieten die Möglichkeit, farbabhängig bedingtes OR-Split, OR-Join, AND-Split und AND-Join zu realisieren.

Das Entscheidungselement (OR-Split) ist mit einer eingehenden und zwei oder mehreren ausgehenden Transitionen gestaltet. Dabei muss die ganze Farbpalette, welche in der Funktion F (Abbildung 50 [Riab2001]) vorhanden ist, nach dem Entscheidungselement vollständig verteilt sein. Die Funktionen F1...Fm müssen alle Farben der eingehenden Funktion F enthalten, aber eine Farbe darf nur einmal in einer aus den Funktionen F1 bis Fm vorkommen.

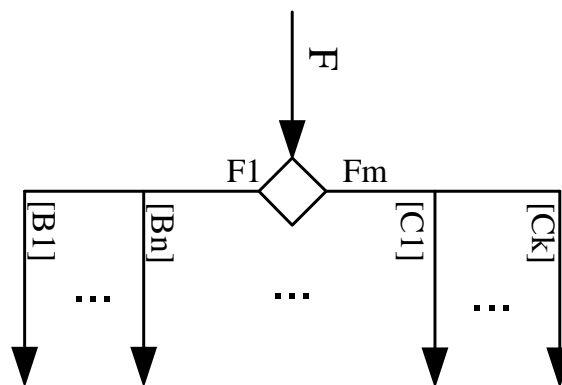


Abbildung 50: Beispiel für einen OR-Split

Das in Abbildung 50 dargestellte OR-Split hat eine eingehende Transition mit der Transitionsfunktion F und mehrere ausgehende Transitionen mit den Transitionsfunktionen von F1 bis Fm. Die Funktionen F1 bis Fm stellen farbabhängige Spezifikationen für Transitionen dar. Dabei können sie noch zusätzlich auf Gruppen mit verschiedenen Transitionsüberwachungsbedingungen (B1..Bn, C1..Ck) aufgeteilt werden.

In Gegensatz zum Entscheidungselement dient ein Vereinigungselement (Or-Join) zur Zusammenführung von Transitionen, welche vorher aufgespaltet wurden (Abbildung 51).

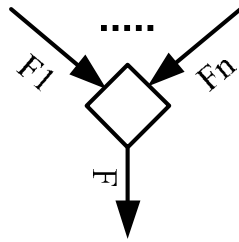


Abbildung 51: Beispiel für einen OR-Join

Die ausgehende Funktion F muss alle Farben von eingehenden Funktionen von F_1 bis F_n enthalten.

Das Verzweigungselement oder AND-Split wird für die Realisierung von Parallelitäten eingesetzt. Dabei können gleiche Farben in mehreren Transitionsfunktionen vorkommen. Der Synchronisationsbalken des Verzweigungselements beinhaltet eine eingehende und mehrere ausgehenden Transitionen, welche alle Farben aus der Funktion der eingehenden Transition aufweisen müssen (Abbildung 52 [Riab 2001]). Dabei existieren für jede Farbe nur die Transitionen, in deren Funktionen sie enthalten ist.

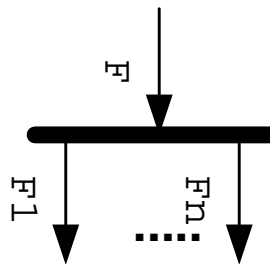


Abbildung 52: Beispiel für einen AND-Split

Das Synchronisationselement oder And-Join fasst mehrere Transitionen zu einer zusammen (Abbildung 53). Die Funktion F der ausgehenden Transition muss alle Farben von allen eingehenden Transitionen beinhalten. Für jede einzelne Farbe, für die keine Abhängigkeitsbedingungen vordefiniert sind, kann dieses Synchronisationselement einzeln ausgeführt werden.

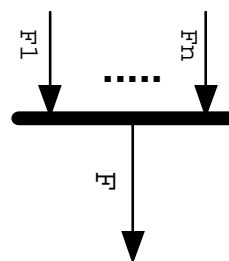


Abbildung 53: Beispiel für einen AND-Join

3.3.5.3 Erweiterung von anderen Elementen im Aktivitätsdiagramm

Die Start- und Endzustände in gefärbten Diagrammen stellen komplexe Elemente dar (Abbildung 54).

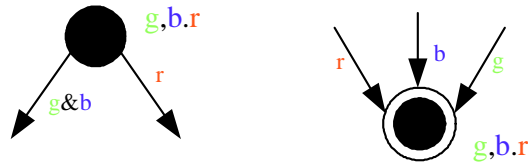


Abbildung 54: Beispiel für einen Start- und einen Endzustand

Ein Startzustand beinhaltet alle Farben, welche für dieses Aktivitätsdiagramm definiert sind, und hat eine oder mehrere ausgehende Transitionen, welche für verschiedene Farben passierbar sind. Die Marken verschiedener Farben können den Startzustand gleichzeitig oder sequentiell und in beliebiger Reihenfolge verlassen. Das Erreichen des Endzustands für eine beliebige Farbe bedeutet, dass das Aktivitätsdiagramm für diese Farbe ausgeführt ist.

Das Verfeinerungskonzept für gefärbte Aktivitätsdiagramme basiert vollständig auf dem Verfeinerungskonzept der Aktivitätsdiagramme der UML (Abbildung 55).

Dabei müssen die Kapazitäten von Unteraktivitäten an die Kapazitätsfunktionen der übergeordneten Aktivitäten angepasst sein.

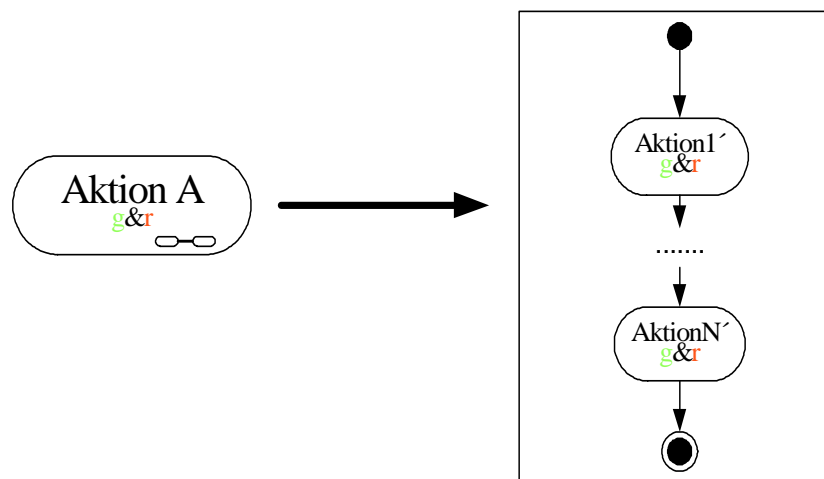


Abbildung 55: Verfeinerter Aktivitätszustand

3.3.3. Berücksichtigung von Zeitbedingungen

In den Aktivitätsdiagrammen der UML sind Zeiten nicht explizit vorgesehen, aber die Einführung von Zeitintervallen kann durch standardisierte Mittel, wie z.B. Zeitereignisse, erfolgen. Dabei modelliert ein Zeitintervall die minimale und maximale Dauer der Ausführung der Aktion. Das Zeitintervall kann für jede Farbe einzeln angegeben werden.

Eine mit einer bestimmten Zeitausführung versehene Aktion wird als Zustand mit Eintrittsaktion und mit einer ausgehenden Transition, an welcher das Zeitintervall notiert ist, modelliert. Beim Eintreten in diesen Aktionszustand wird der Timer für eine bestimmte Farbe mit einem Wert innerhalb des Intervalls (je nach Modellierungszweck wahlweise, die untere oder obere Grenze) gesetzt und bei Übereinstimmung des Timerwertes und des angegebenen Zeitpara-

meters findet dann dieses Zeitereignis statt und der Kontrollfluss für diese Farbe kann den Aktionszustand bei Ausführbarkeit der anderen Bedingungen verlassen.

Das Zeitintervall stellt dabei einen Parametersatz für das Zeitereignis dar (Abbildung 56 [Riab 2001]). Bei für alle Farben gleichem Zeitintervalle wird ein Zeitereignis ohne Farbspezifikation angegeben: after (Tmin, Tmax). Bei unterschiedlichen Zeitintervallen für verschiedene Farben wird das Zeitereignis farbbezogen angegeben: after (Tmin1, Tmax1)/Farbe1; after (Tmin2, Tmax2)/Farbe2.

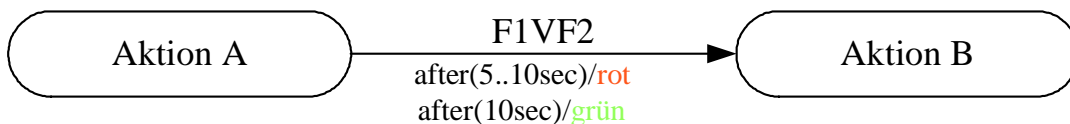


Abbildung 56: Notation von Zeitereignissen (Intervalle oder Werte)

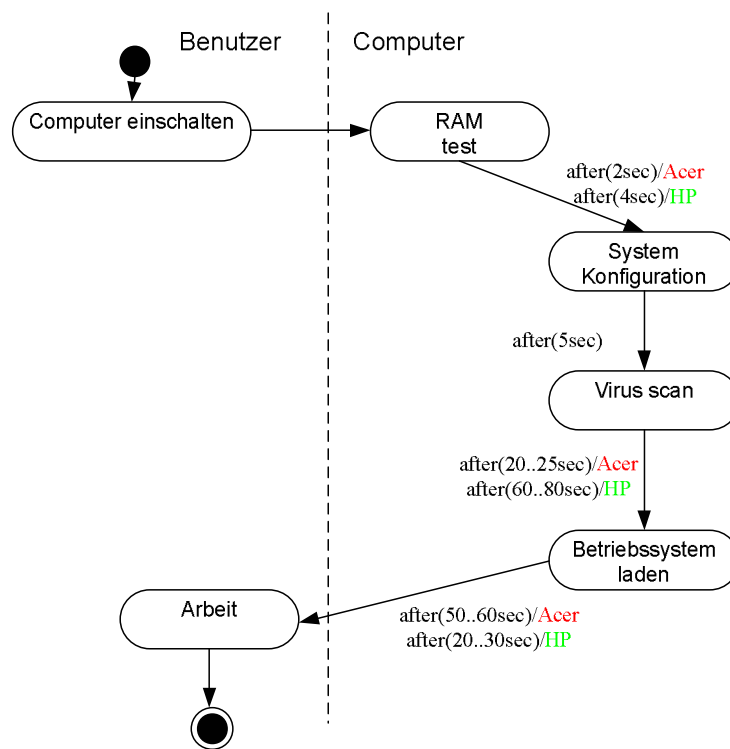


Abbildung 57: Berücksichtigung der Zeit beim Modellieren

In Abbildung 57 [Riab 2001] wird der zeitabhängige Ablauf bei verschiedenen Computertypen (Acer- Computer mit 64-Megabyte RAM (rot) und Hewlett-Packard Computer mit 128-Megabyte RAM (grün)) dargestellt. Beide Rechner sind dazu mit verschiedenen Antiviruserprogrammen ausgestattet. Wegen des RAM-Unterschiedes werden einige Aktionen beim HP-Rechner schneller beendet, dabei gibt es auch zeitliche Unterschiede beim Ablauf der Antiviruserprogramme.

3.3.4. Funktionsweise

Die Funktionsweise von gefärbten Aktivitätsdiagrammen wird hier an Hand eines Beispiels gezeigt. In diesem Beispiel kommen vier gefärbte Aktionszustände und ein Verzweigungs-

element vor (Abbildung 58 (nach [Riab 2001])). Dabei sind an diesem Diagrammausschnitt nur zwei Prozesse, welchen durch die Farben rot und grün unterschieden werden, beteiligt.

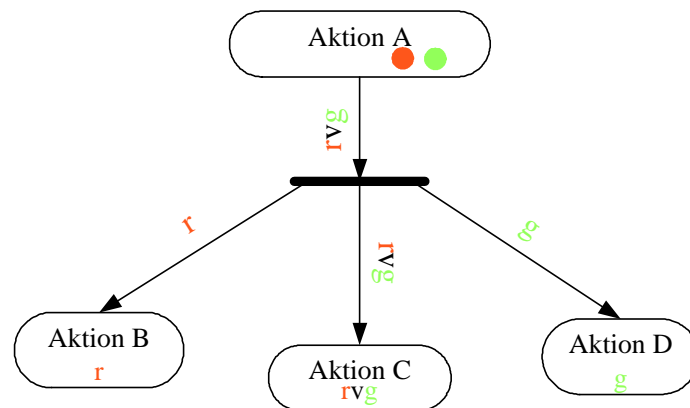


Abbildung 58: Funktionsweise für ein Beispiel eines gefärbten Aktivitätsdiagrammes

Die Aktion A kann für beide Farben ausgeführt werden. Sie kann nur für beide Farben unabhängig verlassen werden. Danach werden die Aktionen B und D entsprechend nur für die rote bzw. grüne Farbe durchgeführt. Parallel ist Aktion C entweder für die rote oder für die grüne Farbe ausführbar, je nachdem, welche Farbe Aktion A gerade durchlaufen hat. Die Transitionsfunktionen sind durch r , rVg , g definiert. Beide Prozesse, welche durch diese Farben gekennzeichnet sind, befinden sich anfangs in Aktion A.

Bei der Ausführung von Aktion A für die rote Farbe wird je eine Marke dieser Farbe in zwei Transitionen erzeugt und diese erscheinen in den Aktionen B und C (Abbildung 59 [Riab 2001]).

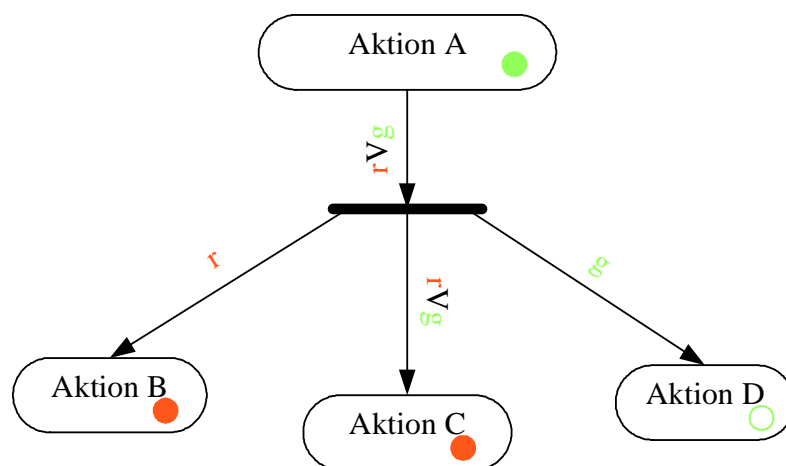


Abbildung 59: Ausführung von Aktion A für die rote Farbe

Solange die Aktion C für die rote Farbe nicht ausgeführt ist, kann die grüne Marke die Aktion A nicht verlassen. Das geschieht erst nachdem die Aktion C für die rote Marke beendet ist und diese die Aktion verlassen hat (Abbildung 60 [Riab 2001]).

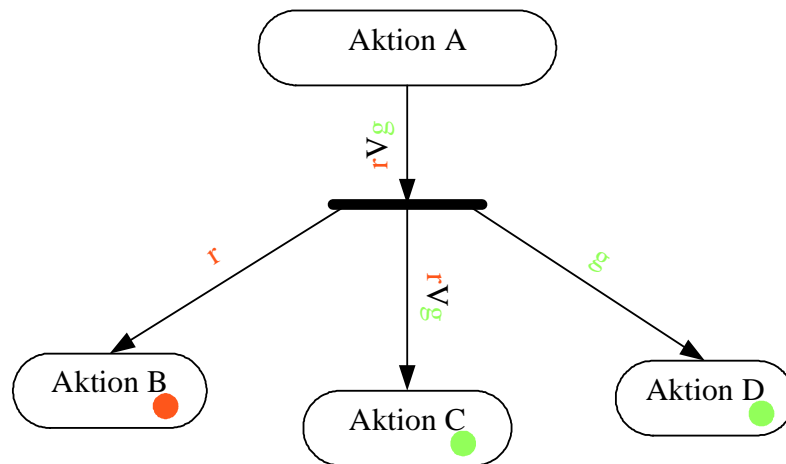


Abbildung 60: Ausführung von Aktion A für die grüne Farbe

Beim Modellieren mit gefärbten Aktivitätsdiagrammen sollen die folgenden Entwurfsregeln beachtet werden. Diese sind eine Weiterentwicklung der Regeln für normale Aktivitätsdiagramme aus [Mühl 98] zu den hier behandelten gefärbten Aktivitätsdiagrammen.

1. Ein Verzweigungselement darf nicht mit einer Vereinigung abgeschlossen werden (Abbildung 61 [Riab 2001]). Bei Verstoß gegen diese Regel werden zwei Marken gleicher Farbe in einem sequentiellen Zweig vorkommen.
2. Nach dem Entscheidungselement darf keine Synchronisation folgen (Abbildung 62 [Riab 2001]), da bei Verstoß gegen diese Regel die Marke nach dem Entscheidungselement nur in eine der Zielaktionen eintreten kann, was zur Nichtschaltfähigkeit des nachfolgenden Synchronisationselementes führt.
3. Alle Marken, die in der Verzweigung aufgespaltet wurden, müssen genau in einer Synchronisation vereinigt werden. Bei Verstoß gegen diese Regel, falls weniger Marken für die Synchronisation benötigt werden, verbleiben Marken in einem oder mehreren Zweigen. Falls mehr benötigt werden, führt das wiederum zur Nichtschaltfähigkeit des nachfolgenden Synchronisationselementes (Abbildung 63 [Riab 2001]).

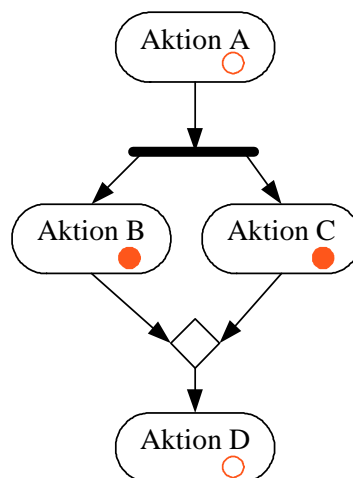


Abbildung 61: Beispiel für ein falsch entworfenes Diagramm (Regel 1)

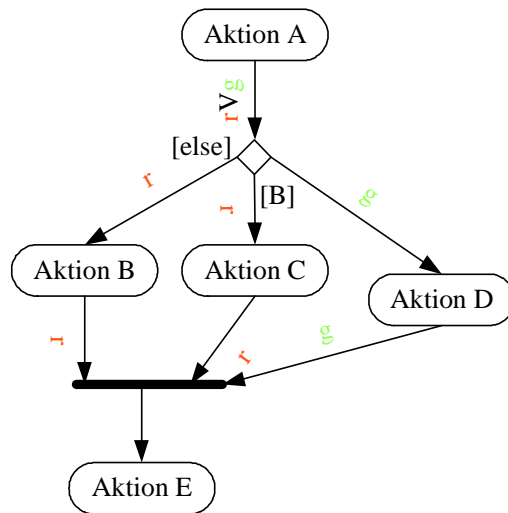


Abbildung 62: Beispiel für ein falsch entworfenes Diagramm (Regel 2)

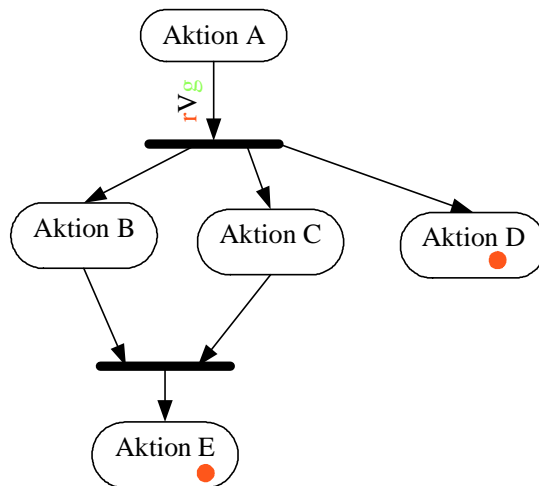


Abbildung 63: Beispiel für ein falsch entworfenes Diagramm (Regel 3, zu viele Marken)

3.4 Gefärbte Sequenzdiagramme

Oft hat man es mit technischen Prozessen zu tun, an denen viele ähnliche Objekte beteiligt sind, z.B. Sensoren. Das System kann dabei z.B. folgendes Verhalten aufweisen: das Warten auf alle Messwerte oder das Warten auf einen von mehreren. Die Einführung von Farben beim Modellieren solcher Situationen in Sequenzdiagrammen bringt die Möglichkeit, übersichtliche und gut strukturierte grafische Darstellungen zu gewinnen. Die Erweiterung eines Sequenzdiagramms um Farben erlaubt das Zusammenfassen von mehreren Darstellungselementen zu einem neuen gefärbten Element, z.B. das Zusammenfassen von mehreren Nachrichten zu einer komplexen Nachricht. Dieses Prinzip entfernt die mehrfach dargestellten Teile aus dem Diagramm, wobei die Parallelität von identischen oder ähnlichen Teilprozessen erhalten bleibt.

3.4.1 Definition

Def. Ein gefärbtes Sequenzdiagramm beschreibt den möglichen gefärbten Nachrichtenaustausch, welcher zwischen farbigen Objekten stattfindet. Diese entstehen durch die Faltung von mehreren Objekten mit ähnlichem Verhalten, die zu einer Klasse oder verschiedenen Unterklassen einer Klasse gehören. (Abbildung 64 [Ambe 2001]). Das gefärbte Sequenzdiagramm besteht aus:

- Komplexen Instanzen, welche ein gemeinsames Objektsymbol (Instanzierungskopf) und eine Lebenslinie für verschiedene, aber ähnliche Objekte haben;
- verschiedenen komplexen Nachrichtenarten, welche mit optionalem Bezeichner und/oder Parametern versehen sind und vom synchronen oder asynchronen Typ sind sowie nach Methodenaufrufen mit Rückantworten unterschieden werden;
- komplexen Instanzerzeugungspfeilen, die gleichzeitige Erzeugung von mehreren Objekten erlauben;
- komplexen Objektzerstörungen;
- komplexen bedingten Nachrichten und alternative Lebenslinien;
- Iterationsmarken und Wiederholungsschleifen für mehrfaches Auftreten einer komplexen Nachricht bzw. eines Abschnittes in einem Sequenzdiagramm;
- auf gefärbte Nachrichten angepassten Zeitkonstrukten.

Gefärbte Sequenzdiagramme werden genauso wie Sequenzdiagramme der UML durch zwei Darstellungsdimensionen repräsentiert, wobei die vertikale Dimension einen zeitlichen Ablauf zeigt und in der horizontalen Dimension die betrachteten Objekte unterschieden werden. Die Erweiterung der Sequenzdiagramme der UML zu gefärbten Sequenzdiagrammen besteht in dem Vorhandensein mehrerer Objekte in einem Instanzierungskopf (Objektsymbol), welche durch verschiedene Farben unterschieden werden und durch entsprechende Marken verkörpert sind. Dazu kommen noch farbabhängige Nachrichten, welche bestimmte Sender mit bestimmten Empfängern in Verbindung setzen (Abbildung 64 [Ambe 2001]).

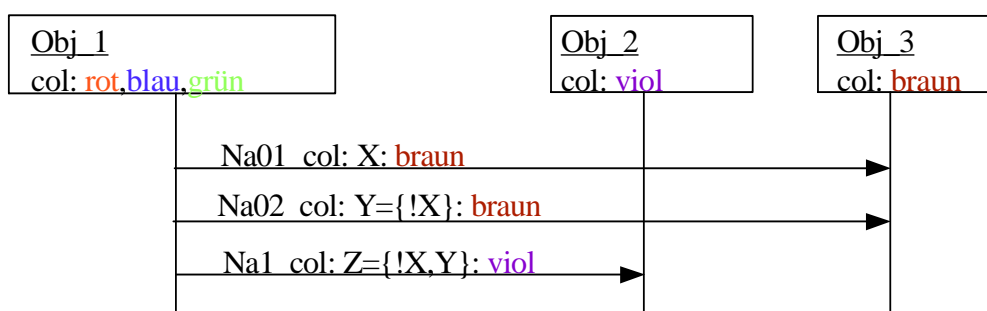


Abbildung 64: Beispiel für ein gefärbtes Sequenzdiagramm

3.4.2 Syntaxerweiterung

Gefärbte Instanzen werden wie in Sequenzdiagrammen der UML durch Rechtecke (Objektsymbol) mit dem Namen der Instanz abgebildet (Abbildung 65). Da dieser Name nicht mehr eine Instanz, sondern eine Gruppe von verhaltensähnlichen Instanzen repräsentiert, wird zur Unterscheidung dieser Objekte von einander eine zusätzliche Zeile mit dem Schlüsselwort

colour (abgekürzt *col*) eingeführt, nach welchem eine Reihe von unterschiedlichen Farben (Typen) folgt.

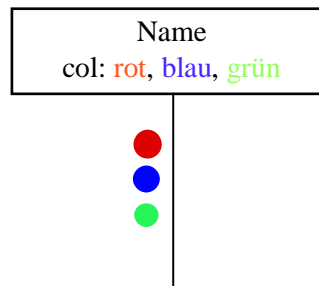


Abbildung 65: Repräsentation von gefärbten Objekten

Die Lebenslinie ist jetzt für mehrere, durch die Farben unterschiedene Objekte aktuell. Dabei stellt die aktuelle Position der Lebenslinie für eine Farbe einen gefärbten Zustand dar. Es lassen sich damit n ähnliche Objekte auf n verschiedene Objekte abbilden, welche unterschiedliche Nachrichten senden können.

Die Nachrichten werden in gefärbten Diagrammen zusätzlich mit farbabhängigen Informationen versehen, welche explizit Sender und Empfänger definieren und dabei eine Farbkonvertierungsfunktion ähnlich wie bei gefärbten Petri-Netzen darstellen. Eine solche Nachricht wird senderseitig von einer oder mehreren Farben erzeugt und entsprechend von den korrespondierenden Farben der Empfängerseite übernommen. Diese Farbinformation wird durch folgende Notationsweise realisiert: nach dem Schlüsselwort *col*: folgen durch Doppelpunkt getrennte „Senderfarbe“ und „Empfängerfarbe“. Dabei muss als Bedingung zur Nachrichtversendung die Übereinstimmung der Farbe des aktuellen Zustandes des Objekts mit der definierten Farbe für die Senderseite im Nachrichtentupel erfüllt sein.

Die Einführung von Variablen bei der Beschreibung der beteiligten Sender und Empfänger bringt mehr Variabilität und Übersichtlichkeit in die Diagramme. Das Tupel von Sender- und Empfängerfarben kann dabei nicht nur skalar sein, sondern anstelle von skalaren Größen auch Vektoren bzw. Felder enthalten. Als Bedingung zur Verwendung von Vektoren wird die gleiche Größe der Sender- und Empfängerfelder vorausgesetzt. Um Konstanten und Variablen unterscheiden zu können, werden Farbkonstanten mit Kleinbuchstaben und Variablen mit Großbuchstaben gekennzeichnet. Als weitere Syntax zur Beschreibung der farbabhängigen Nachrichten wird nach [Ambe 2001] folgendes festgelegt:

- „{ }“: In geschweiften Klammern folgt eine Aufzählung von erlaubten Farbvariablen.
- „=“: Das Gleichheitszeichen führt zu einer Einschränkung von zulässigen Werten einer Farbvariable.
- „!“: Das Ausrufungszeichen definiert den Gültigkeitsbereich für allen Farben, außer den nach diesem Zeichen aufgeführten Farben.
- „*“ oder []: Ein führender Stern oder rechteckige Klammern geben eine Hinweis auf die Benutzung von Farbvariablen als Vektoren, dabei wird auf die einzelnen Elemente des Vektors $*X$ mit $X[1]..X[n]$ zugegriffen. Zum Beispiel:

$$*X = \{\text{rot}, \text{grün}\}$$

$$*X = [\{\text{rot}\}, \{\text{blau}\vee\text{grün}\}]$$

Im ersten Fall beschränken sich alle Elemente auf rot und blau. Im zweiten Fall muss das erste Element rot sein und das zweite entweder grün oder blau.

- ANY: Eine vordefinierte Farbvariable, wobei das farbige Objekt, das zuerst diesen Zustand erreicht, diese Nachricht generieren oder konsumieren wird. ANY wird jedes Mal als eine neue Farbvariable benutzt, sie ist nicht gebunden und kann bei jedem neuen Auftreten jedes Mal ein neuer Farbwert aufweisen.
- ALL: Eine vordefinierte Farbvariable, welche zeigt, dass alle Farben der sendenden oder empfangenen Instanz die Nachricht erzeugen oder konsumieren können (Abbildung 66 [Ambe 2001]).

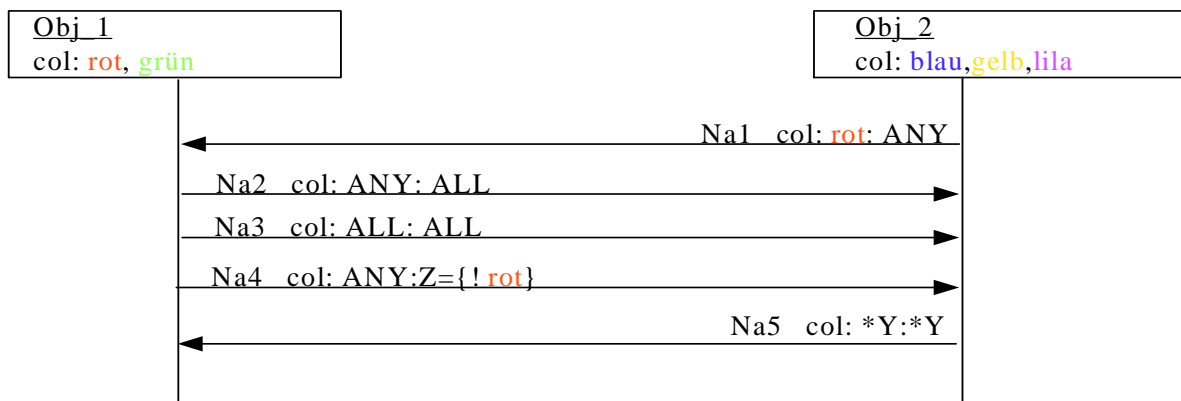


Abbildung 66: Farbabhängige Nachrichtentypen

Die Farbvariablen gelten für den gesamten Bereich des Sequenzdiagramms und können dabei nur einen Farbwert (außer der vordefinierten Variablen ANY) annehmen. In der Beschreibungsliste von Variablen können alle auszuführenden oder verbotenen Werte angegeben werden, z.B.

„X darf rot oder grün annehmen“ wird als $X = \{\text{rot} \vee \text{grün}\}$,
 „X darf alles außer blau annehmen“ wird als $X = \{\text{!blau}\}$

bezeichnet.

Wenn man die Menge von Farben auf der Senderseite durch C1 (Color Menge 1) bezeichnet und die Menge von Farben auf der Empfängerseite durch C2, sind dabei die Mengen C1' und C2' Objektlisten für verschickte und empfangene Nachrichten [Rang 01]. Dann wird für die Objektfarblisten „C1“, „C1'“, „C2“ und „C2'“ gelten:

Sender-/Empfängerseite:

- $C1 = \{c_{11}, c_{12}, \dots, c_{1n}\} \neq \emptyset$
- $C2 = \{c_{21}, c_{22}, \dots, c_{2n}\} \neq \emptyset$
- $C1 \cap C2 = \emptyset$, was zeigt, dass die Mengen C1 und C2 disjunkte Mengen sind und *keine Selbstinteraktionen* vorliegen.

Senderseite:

- $C1' \subseteq C1$, es gilt: $C1' \neq \emptyset$
 - Fall 1: $C1' \subset C1$
 - Fall 2: $C1' = C1$

Empfängerseite:

- $C2' \subseteq C2$, es gilt: $C2' \neq \emptyset$
 - Fall 1: $C2' \subset C2$
 - Fall 2: $C2' = C2$

Es ergeben sich dabei folgende Varianten:

- 1.Fall: $C1' = C1$ und $C2' = C2$
- 2.Fall: $C1' \subset C1$ und $C2' \subset C2$
- 3.Fall: $C1' = C1$ und $C2' \subset C2$
- 4.Fall: $C1' \subset C1$ und $C2' = C2$

Als weitere Syntax zur Beschreibung der verschiedenen Kombinationsmöglichkeiten wird folgendes vorgesehen:

- $(c1\&c2)$, $(c1,c2) = c1$ UND $c2$, dabei entspricht das Kommazeichen einer UND-Verknüpfung
- $(c1\vee c2) = c1$ ODER $c2$

3.4.3. Zeitbewertung

Die Zeitkonstrukte in gefärbten Sequenzdiagrammen sind genauso wie in gefärbten Zustands- und Aktivitätsdiagrammen mit Ereignissen verbunden. In normalen Sequenzdiagrammen der UML werden diese Zeitbedingungen neben der Lebenslinie bei dem Senden oder Empfangen einer Nachricht notiert. Diese Zeitbedingungen stellen die einzuhaltende Zeitspanne zwischen zwei Nachrichten dar und werden zusammen mit den Namen dieser Nachrichten notiert. In gefärbten Sequenzdiagrammen können diese beiden Nachrichten mehrmals durch verschiedene Farben verschickt oder empfangen werden. Die Zeitbedingungen werden dabei für alle für diese Nachrichten relevanten Farben gelten. Bei Angabe von farbübergreifenden Zeitbedingungen werden diese Farben in Klammern berücksichtigt. Dabei ist die Syntax der Angabe der Farben genauso wie bei den Farbangaben von Nachrichten: Es können Farbkonstanten, Farbvariablen mit und ohne Einschränkungen, Farbvektoren und die vordefinierten Variablen ANY und ALL verwendet werden.

In Abbildung 67 [Ambe 2001] sind verschiedene Varianten zur Angabe von Zeitbedingungen in gefärbten Sequenzdiagrammen dargestellt. Im ersten Fall muss der Methodenaufruf für jede Farbe innerhalb von 10 s abgeschlossen sein. Im zweiten Fall dürfen nur weniger als 10 s zwischen dem Aufruf von der ersten Farbe und der Beendigung des Aufrufs der letzten Farbe vergehen. Im letzten Fall muss die erste Methode innerhalb von 10 s nach dem Starten des letzten Methodenaufrufs beendet sein.

In der Erweiterung des beschriebenen Konzeptes ist auch die Verwendung von Zeitintervallen an Stelle der Zeitangaben möglich.

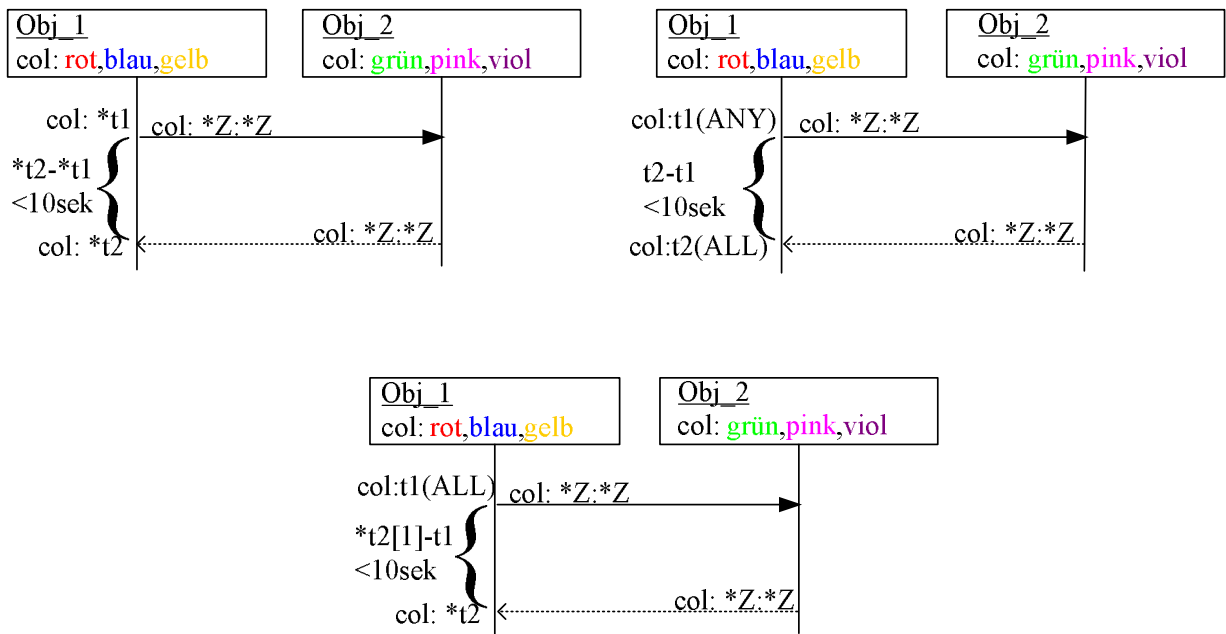


Abbildung 67: Beispielhafte Zeitangaben für gefärbte Sequenzdiagramme

3.4.4 Funktionsweise

Die Funktionsweise von gefärbten Instanzen entspricht im übertragenen Sinne der Funktionsweise von Instanzen der Sequenzdiagramme der UML. Die aktive gefärbte Instanz ist durch ein Rechteck mit dem Verweis auf alle geltenden Farben gekennzeichnet und ihre Lebenslinie verläuft senkrecht, für jede Farbe unabhängig voneinander, nach unten (Abbildung 68).

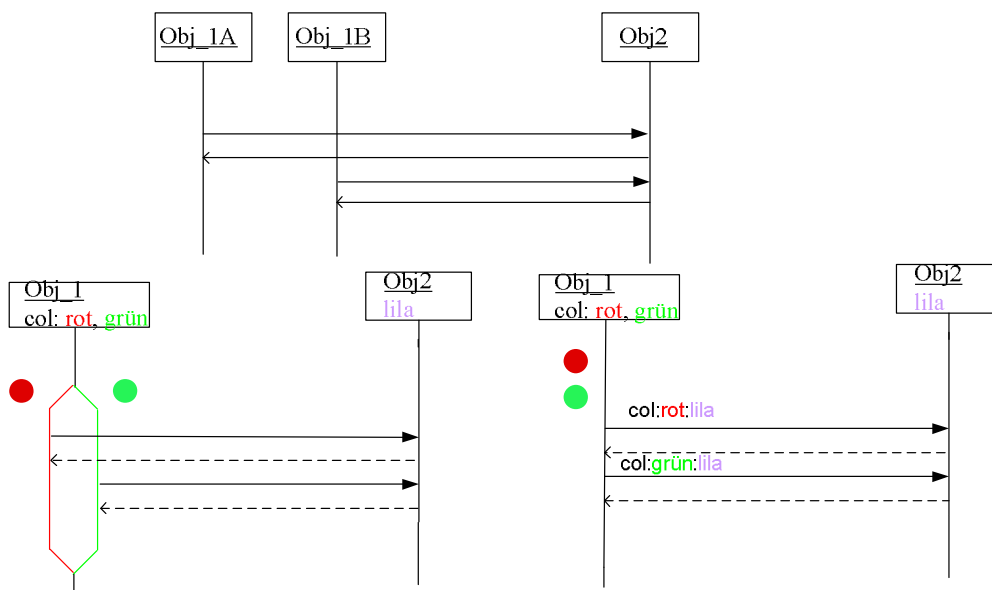


Abbildung 68: Beispielhafte Verläufe von Lebenslinien für zwei verschiedene Farben

Die farbabhängigen Nachrichten erlauben dabei, die Aufspaltung von Lebenslinien zu vermeiden (Abbildung 68). Sollte die Aufspaltung aber verwendet werden, wird die im weiteren

beschriebene farbabhängige Alternative genutzt, die eine farbabhängige Parallelverarbeitung modelliert. Die Marken wandern die Lebenslinie entlang nach unten und kennzeichnen damit aktuelle lokale Zustände, welche die entsprechenden Objekte annehmen können. Zu jedem Zeitpunkt kann aber nur das einmalige Senden oder Empfangen von Nachrichten pro Farbe vorgegeben sein. Erreicht die Marke diesen Zeitpunkt, geschieht der Nachrichtenaustausch. Die Erweiterung um Farben gilt gleichermaßen für alle Nachrichtentypen, deshalb geht es im weiteren allgemein um „Nachrichten“. Farbabhängige Nachrichten können als Werte Konstanten, Variablen oder die vordefinierten Variablen ANY oder ALL aufweisen. Dabei wird folgendes Verhalten vordefiniert:

Für Farbkonstanten:

- Auf der Senderseite: Die Nachricht wird nur einmal bei Erreichen der in dieser Nachricht aufgezählten Farben von diesem Zustand der Lebenslinie erzeugt.
- Auf der Empfängerseite: Die Nachricht wird nur von den im zweiten Teil des Tupels angegebenen Farben angenommen. Für alle anderen Farben geschieht keine Nachrichtenannahme an dieser Stelle.

Für Farbvariablen:

- Auf der Senderseite: Wenn eine Variable noch nicht an bestimmte Farben gebunden ist, wird sie beim ersten Auftreten im Sequenzdiagramm mit der Farbe, welche zuerst den Zustand des entsprechenden Verschickens der Nachricht mit dieser Variablen erreicht, belegt.
- Auf der Empfängerseite: Für eine schon gebundene Variable wird die Nachricht nur von der in dieser Variablen gebundenen Farbe übernommen. Beim ersten Auftreten einer Variablen auf der Empfängerseite wird sie mit der Farbe gebunden, welche diese Nachricht zuerst bearbeiten kann. Bei einem Konstrukt wie `col:X:X` und noch nicht festgelegter Farbe für die Variable X wird die Farbe schon beim Senden festgelegt (Abbildung 69 (überarbeitet nach [Ambe 2001])).

Für ANY:

- Auf der Senderseite: In diesem Fall wird die Nachricht einmalig durch die Farbe, die zuerst diesen Zustand erreicht, generiert.
- Auf der Empfängerseite: Die Nachricht wird einmalig durch die Farbe, die zuerst diesen Zustand erreicht, empfangen.

Für ALL:

- Auf der Senderseite: In diesem Fall sind alle Farben für das Verschicken dieser Nachricht verantwortlich. Deshalb wird sie erst verschickt, wenn alle Farben diesen Zustand erreichen.
- auf der Empfängerseite: Die Nachricht wird von allen Farben empfangen.

Die Objekterzeugung und Objektzerstörung verläuft in gefärbten Diagrammen analog zu Sequenzdiagrammen der UML, nur können dabei gefärbte Objekte erzeugt werden. Es können in bereits vorhandenen gefärbten Objekten weitere Farben erzeugt werden. Bei Zerstörung von einem gefärbten Objekt können einzelne Farben zerstört werden. Das Objekt wird nach dem Erreichen des Zerstörungssymbols durch die letzte mögliche Farbe vollständig zerstört. Die übliche Syntax für Bedingungen des Sequenzdiagramms der UML für die Erzeugung von Nachrichten bleibt dabei erhalten und wird wie oben beschrieben farbabhängig erweitert.

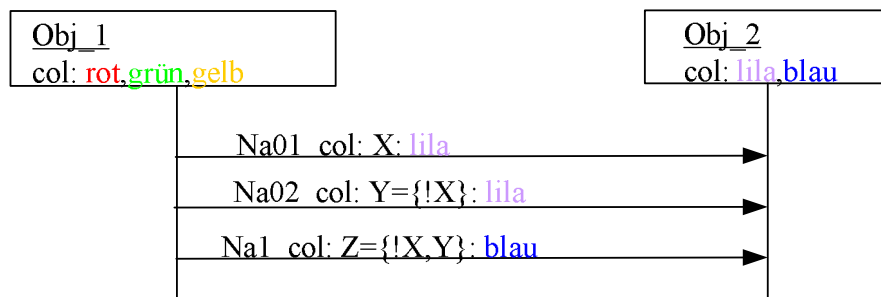


Abbildung 69: Beispiel zur Funktionsweise

Die Verwendung von alternativen Lebenslinien, welche in Sequenzdiagrammen der UML enthalten sind, ist für gefärbte Sequenzdiagramme auch für unterschiedliche Fälle von Farben von Bedeutung (Abbildung 70 [Ambe 2001]). Dabei werden die Farben als Bedingungen angegeben. Die Aufspaltung von Lebenslinien in gefärbten Diagrammen wird dabei aber nicht wie in ungefärbten Sequenzdiagrammen als Alternative genutzt, sondern als parallele Verarbeitung von farbunterschiedlichen Objekten. Die Aufspaltung von Lebenslinien dient dabei zur Verbesserung der Übersichtlichkeit.

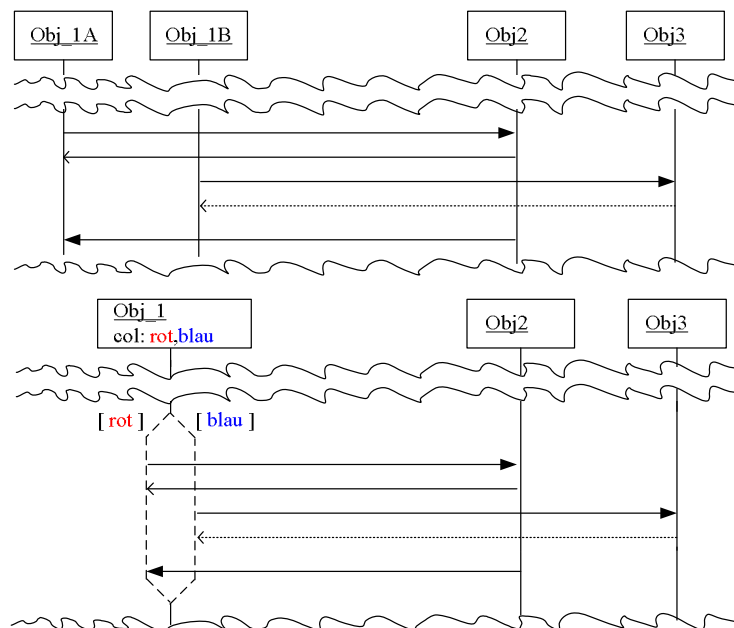


Abbildung 70: Beispiel zur Lebenslinienaufspaltung

Die Iterationsverfahren und Wiederholungsschleifen werden analog zu Sequenzdiagrammen der UML genutzt, wobei diese, wie soeben bei der Alternative beschrieben, farbabhängig sein können.

Vergleich von UML-Standard-Diagrammen mit Gefärbten Diagrammen

In diesem Kapitel wird ein Modell eines Beispielsystems mit gefärbten Aktivitäts-, Zustands- und Sequenzdiagrammen beschrieben und mit UML-Standard-Diagrammen verglichen. Schon bei kleineren Projekten werden normale UML-Diagramme ziemlich schnell zu groß und unübersichtlich. In diesem Kapitel wird am Beispiel des Messsystems, welches in 2.3.2 ausführlich beschrieben ist, dargestellt, wie bei der Lösung von realen Problemen der Einsatz der in dieser Arbeit entwickelten gefärbten Diagramme hilft, die Anschaulichkeit der Modelle zu verbessern. Dabei werden die redundanten Teile von Grafiken entfernt, um den Überblick über das gesamte Projekt zu ermöglichen und die Lesbarkeit der Diagramme zu verbessern. Das Messsystem wird dabei zum Vergleich mit Hilfe von normalen und von Gefärbten Diagrammen modelliert.

4.1. Gefärbtes Aktivitätsdiagramm

In Abbildung 71 (überarbeitet nach [Gren2003]) ist ein normales Aktivitätsdiagramm der UML, welches die Funktionsweise des Messsystems des Laser-Schneidetisches modelliert, dargestellt. Wenn man diesem Diagramm folgt, kann man hier folgenden Ablauf beobachten: Die ersten Aktionen stellen die Läuferbewegungen dar, danach werden Aktionen mit Signal-erzeugung, genauer von Sinus- und Kosinussignalen für die verschiedenen Richtungen: X, Y1, Y2, ausgeführt. Dann erfolgen das Lesen von den Normierungsdaten aus dem Speicher und die Bildung von maximalen und minimalen Werten für X, Y1, und Y2 mit nachfolgender Normierung dieser Daten. Als nächster Schritt werden die Quadranten für X, Y1 und Y2 bestimmt und es wird die Periodenzahl berechnet. Die Normierungsdaten können jetzt aktualisiert werden. Zum Schluss erfolgen die Bildung der aktuellen X- und Y-Werte und die Berechnung der möglichen Verdrehung φ , welche aus den abweichenden Werten von Y1 und Y2 ermittelt werden kann. Die aktuellen X- und Y-Werte und φ werden abgespeichert.

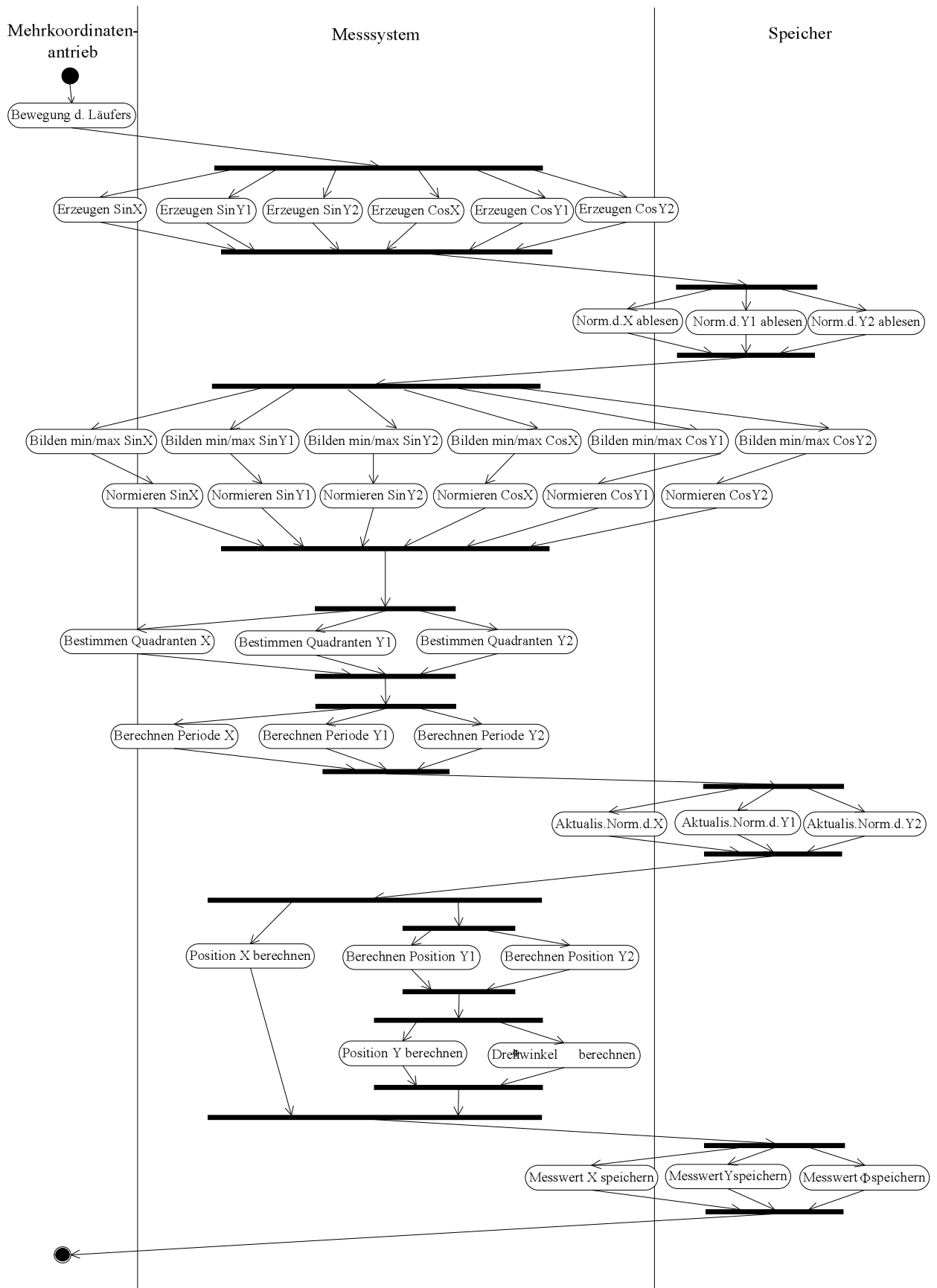


Abbildung 71: Aktivitätsdiagramm entsprechend UML zur Modellierung des Messsystems

An diesem Diagramm kann man sehen, dass viele Prozesse parallel laufen und dabei ähnliches Verhalten haben. Deshalb weist dieses Diagramm viele redundante Teile auf.

In Abbildung 72 ist ein gefärbtes Aktivitätsdiagramm als Modell des Verhaltens dieses Messsystems dargestellt. Dabei kann man das gefärbte Diagramm hier als eine besondere Form der Hierarchiebildung auffassen, welches wesentliche Vorteile bezüglich der Deutlichkeit und Übersichtlichkeit gegenüber normalen Verfahren zur Hierarchiebildung der UML, soweit überhaupt bei den einzelnen Diagrammen vorhanden, aufweist. Bei dieser Darstellungsart gibt es keine Unteraktivitätszustände, was zur besseren Verständlichkeit von der Funktionsweise führt.

Als Farbvariablen werden in diesem Beispiel drei bestimmte Typen X, Y1, Y2, welche die Koordinaten der X-, Y1- und Y2-Achsen darstellen, verwendet. Die Aktionsausdrücke weisen zusätzlich auf mögliche Belegungen dieser Aktionen – beschrieben durch die Kapazität der Aktionen. Die Transitionsfunktionen legen fest, für welche Typen diese Übergänge ausführbar sind. Die Funktionsweise bleibt wie bei den normalen Aktivitätsdiagrammen der UML erhalten. Es werden dabei parallele Aktivitäten zu einer gefärbten Aktivität gefaltet, wie im Beispiel die Sinus- und Kosinuserzeugung und Datennormierung. Bei erhaltener Funktionalität besitzt dieses Diagramm wesentlich weniger Aktionszustände in der grafischen Darstellung.

4.2. Gefärbtes Zustandsdiagramm

In Abbildung 73 [Gren 2003] ist ein Zustandsdiagramm der UML, welche die Funktionsweise des eben verwendeten Messsystems des Laser-Schneidetisches modelliert, dargestellt. Dieses Diagramm beschreibt folgende Schritte: Der erste Zustand stellt die Läuferbewegung dar, danach werden die Zustände mit der Signalerzeugung der Sinus- und Kosinussignale für die verschiedenen Richtungen, X, Y1, und Y2, erreicht. Dann folgt das Lesen der Normierungsdaten aus dem Speicher und die Bildung von maximalen und minimalen Werten für X, Y1 und Y2 mit nachfolgender Normierung dieser Daten. Als nächster Schritt werden die Quadranten für X, Y1 und Y2 bestimmt und Periodenzahl berechnet. Die Normierungsdaten können jetzt aktualisiert werden. Zum Schluss erfolgen die Bildung der aktuellen X- und Y-Werte und die Berechnung der möglichen Verdrehung ϕ , welche aus den unterschiedlichen Werten von Y1 und Y2 ermittelt werden kann. Die aktuellen X- und Y-Werte und ϕ werden abgespeichert.

Dieses Diagramm enthält sehr viele parallele Zustände, welche praktisch äquivalente Zustände für verschiedene Variablen darstellen, was zu mehreren redundanten Teilen im Diagramm und zu schlechter Lesbarkeit des Diagramms führt.

In Abbildung 74 ist ein gefärbtes Zustandsdiagramm, welche das gleiche Messsystemverhalten modelliert, gezeigt. Die bessere Übersichtlichkeit und Deutlichkeit dieses Diagramms wird durch Einführung von gefärbten Zuständen (Faltung von parallelen Zuständen, wie Sinus- und Kosinuserzeugung, Datennormierung und ähnlichen) erreicht. Dabei fehlen hier die so genannten zusammengesetzten Zustände, was die Verständlichkeit der Funktionsweise des Messsystems erheblich verbessert.

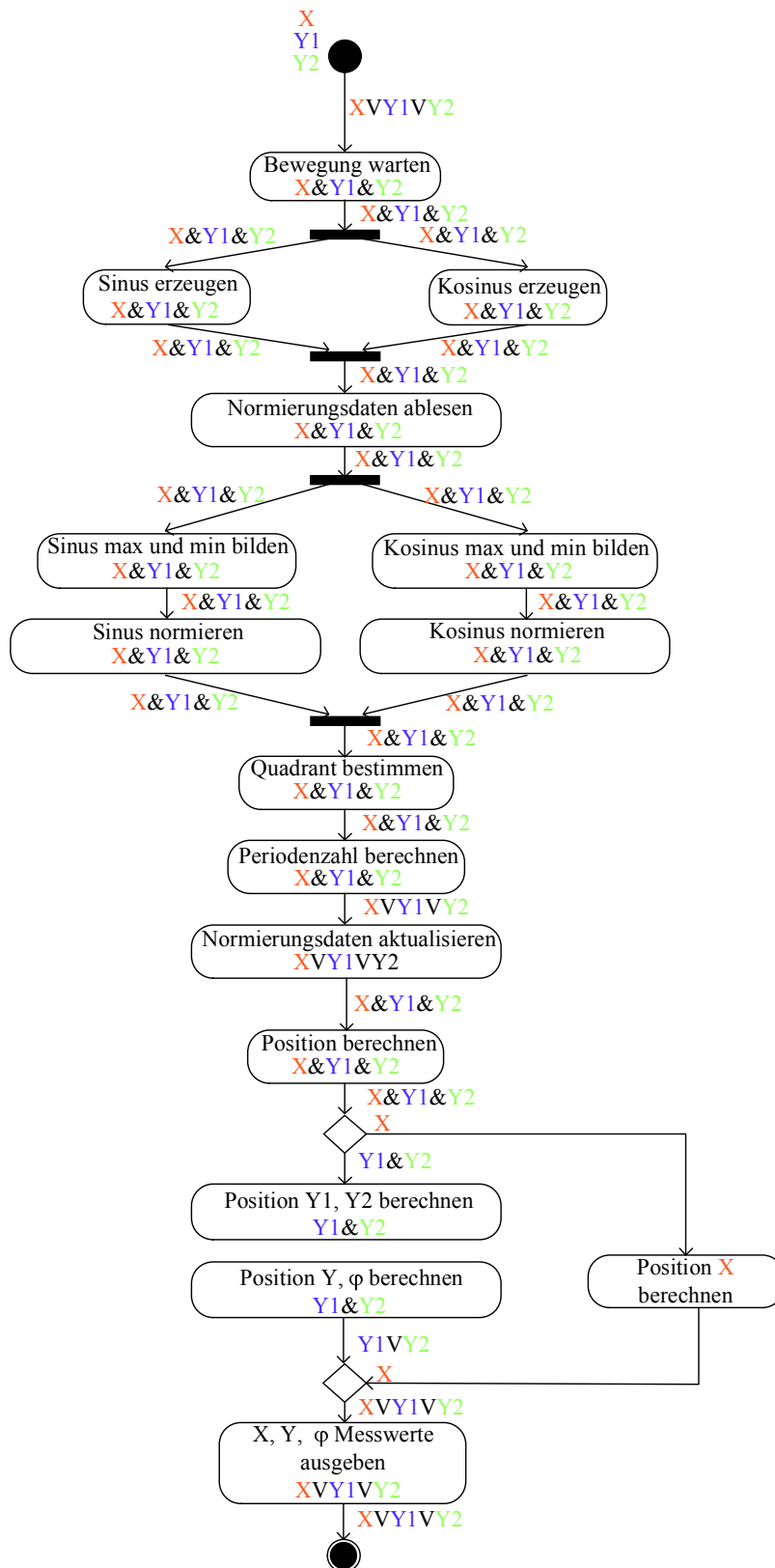


Abbildung 72: Gefärbtes Aktivitätsdiagramm zur Modellierung des Messsystems

Als Farbvariablen werden hier drei bestimmte Farben, X, Y1 und Y2, welche die Koordinaten der X-, Y1-, Y2-Achsen darstellen, verwendet. Die Kapazitätsfunktionen der Zustände verweisen zusätzlich auf mögliche Belegungen dieser Zustände. Die Transitionsfunktionen legen fest, für welche Farben diese Übergänge vollziehbar sind. Die Funktionsweise bleibt wie bei dem normalen Zustandsdiagramm der UML erhalten.

Das Diagramm ist etwas vergrößert und nur zum anschaulichen Überblick dargestellt. Das in Abbildung 74 [Baru 2002] beschriebene gefärbte Zustandsdiagramm fasst mehrere zusammengefaltete Zustände zusammen. In diesem gefärbten Zustandsdiagramm werden folgende Ereignisse und Überwachungsbedingungen zur Steuerung von Zustandsübergängen eingeführt:

- Takt-Ereignis, da das gesamte Messsystem durch Takt-Signale gesteuert wird. Dieses Ereignis wird entweder für alle Farbvariablen X, Y1, Y2 oder nur für bestimmte, dort gültige, festgelegt. Das wird durch den Funktionsausdruck bestimmt.
- Systemfehler-Ereignis. Durch dieses Ereignis werden Fehlersituationen, z.B. außerhalb der Grenzen liegende Werte des Luftdrucks oder der Ströme der aerostatischen Führung, die zum Stoppen des ganzen Systems führen, notiert.

Als Überwachungsbedingung wird [Bew=true] zum Verlassen des Wartezustands definiert.

Die Variable *Pos* bestimmt die Position des Läufers an einer Achse und kommt in dem Zustand „Berechnung der Position“ vor. Dabei wird *Pos* wie folgt berechnet:

$$Pos = 0,04 * (PER + (\arctan(\sin/Kos) / 2 * \pi)),$$

wobei PER die Periodenzahl an der entsprechenden Achse (X, Y1, Y2) ist.

Die durchschnittliche Position Y des Läufers in Y-Richtung und ΔY als mögliche Abweichung werden in dem Berechnungszustand für Y und ΔY folgendermaßen berechnet:

$$Y = (Y1 + Y2) / 2, \Delta Y = Y1 - Y2 / 2.$$

Die Funktionsweise ist ziemlich anschaulich in diesem Diagramm dargestellt, wobei das gleiche Verhalten wie im normalen Zustandsdiagramm der UML beschrieben wird.

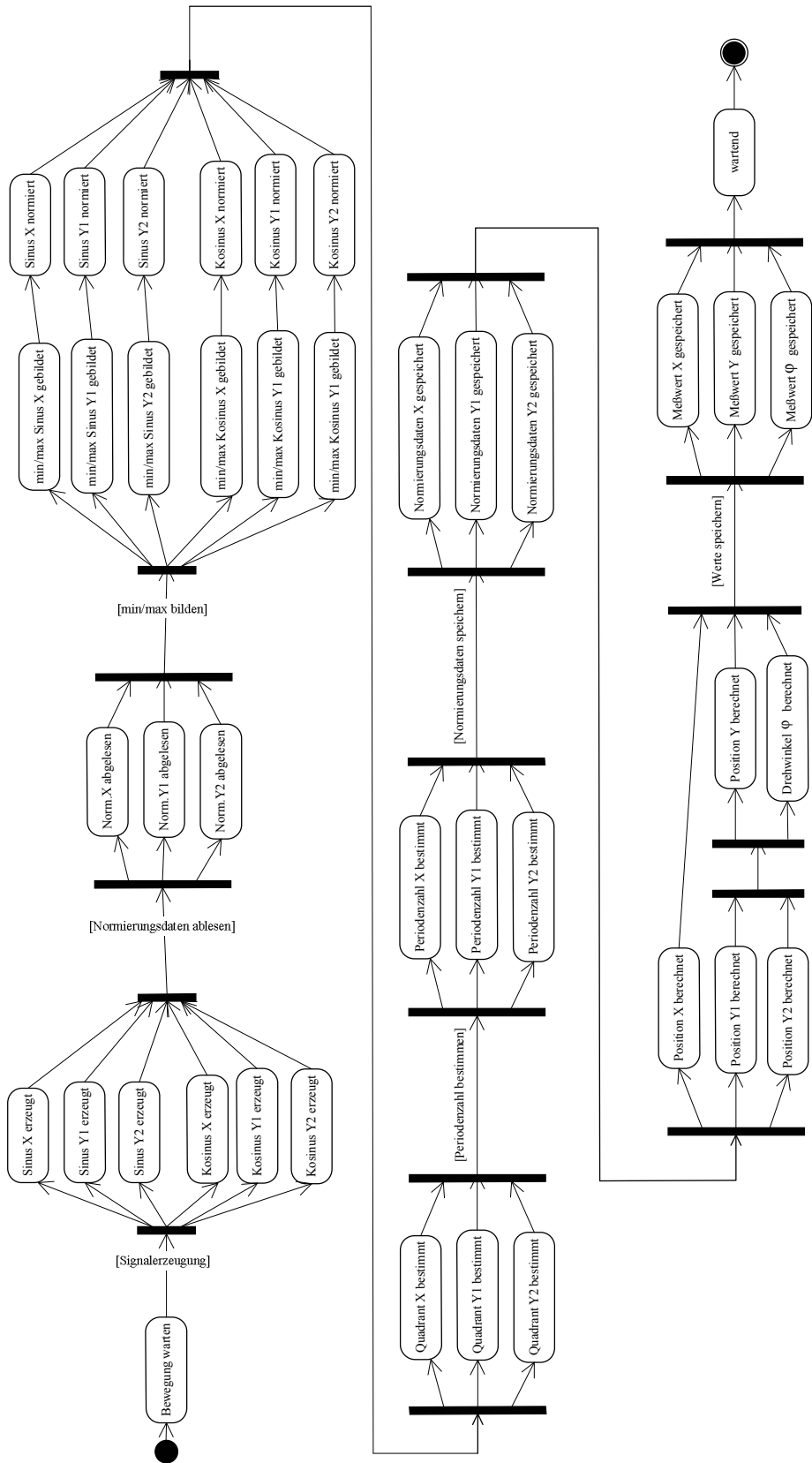


Abbildung 73: Zustandsdiagramm nach UML zur Modellierung des Messsystems.

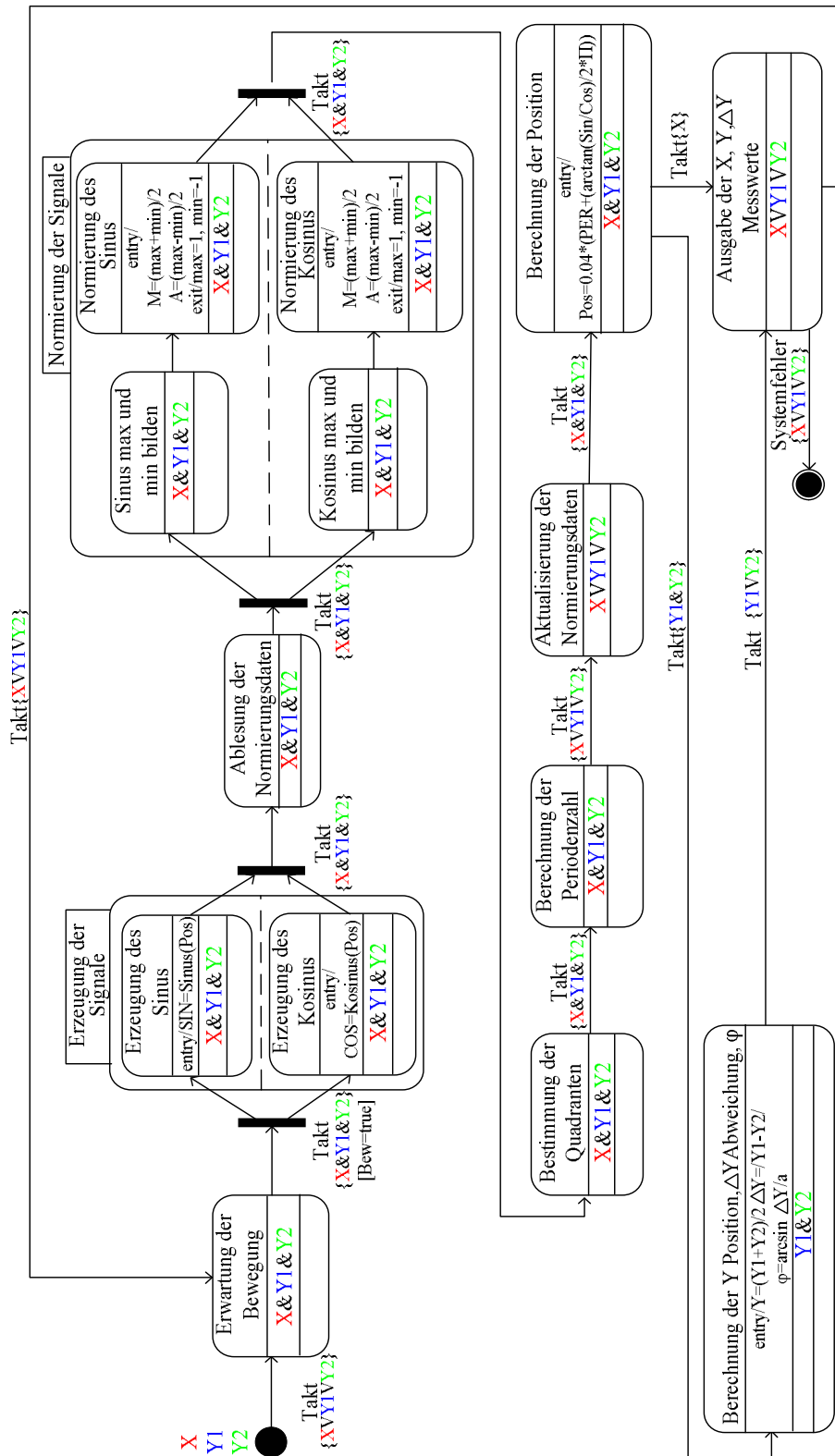


Abbildung 74: Gefärbtes Zustandsdiagramm zur Modellierung des Messsystems.

4.3. Gefärbtes Sequenzdiagramm

Abbildung 75 [Baru 2002] zeigt ein UML-Sequenzdiagramm des Beispiels Messsystem, welches aus dem Message Sequence Chart [Döri 02] abgeleitet ist. In diesem Sequenzdiagramm sind folgende Abkürzungen vorhanden:

$$M=(\max+\min)/2,$$

$$A=(\max-\min)/2,$$

wobei M den Mittelwert des Sinus- oder Kosinussignals und A die Amplitude des entsprechenden Signals darstellt. Nach der Berechnung dieses Mittelwerts M und der Amplitude A geschieht die Normierung dieser Daten: das Rücksetzen der Grenzwerte max und min des entsprechenden Signals auf die Initialwerte 1 und -1.

Das in Abbildung 76 [Baru 2002] dargestellte gefärbte Sequenzdiagramm des Messsystems enthält gefärbte Instanzen als eine Zusammenfassung von mehreren Instanzen aus dem vorher beschriebenen Modell. In diesem Beispiel werden die zwei Instanzen, welche den Analog-Digitaler-Wandler und den Speicher beschreiben, zu einer farbigen Instanz zusammen gefasst. Und die Instanzen X, Y1, Y2, welche die Positionen und die mögliche Verdrehung für die verschiedenen Richtungen modellieren, werden zu einer gefärbten Instanz mit dem Namen „Position“ zusammen gefasst. Durch Einführung von unterschiedlich gefärbten Typen werden die verschiedenen Objekte X, Y1, Y2 definiert. Statt realer Farben sind hier die Namen ursprünglicher Instanzen als symbolische Farbbezeichner gewählt. Das entstandene gefärbte Sequenzdiagramm beschreibt dieselbe Funktionalität wie das normale Sequenzdiagramm der UML. Es kommt aber mit erheblich weniger Elementen aus, was zu besserer Übersichtlichkeit führt.

In Abbildung 77 [Ambe 2001] ist ein weiteres ungefärbtes Sequenzdiagramm zu Modellierung der Läuferbewegung im Messtisch gezeigt. Es wird hier die Bewegung des Läufers beschrieben. Als Ausgabedaten entstehen die Sinus- und Kosinus-Werte der X- und der beiden Y1- und Y2-Richtungen. Dazu werden folgende fünf Parameter, die die Bewegungen nach oben, unten, rechts und links und das Stoppen der Bewegung beschreiben, definiert: HO, RU, LI, RE und ST. Da man in der Realität immer mit unerwünschten Verschiebungen rechnen muss, werden hier auch zusätzlich drei mögliche Störungen simuliert. Der Normwert wird ständig verändert, so dass der Mittelpunkt von Null abweicht. Dieser ist in dem Diagramm durch RndNorm dargestellt. Durch RndAmp werden Amplitudenveränderungen berücksichtigt. Und mit Rndzeit werden die Werte von Sinus und Kosinus mit zeitlichen Verschiebungen belegt.

Die in Abbildung 78 [Ambe 2001] gezeigten Instanzen X, Y1, Y2 erhalten genaue Positionsangaben, dabei ist ihr Verhalten ziemlich übereinstimmend, da sie alle in regelmäßigen Abständen abhängig von den Richtungsangaben ihre Koordinaten ändern. Die Instanzen X_mes, Y1_mes, und Y2_mes, welche genau die gemessenen Werte übermitteln, verhalten sich fast identisch.

Zur Darstellung dieses Modellierungsbeispiels als ein gefärbtes Sequenzdiagramm werden die Instanzen X, Y1, Y2 und die Instanzen X_mes, Y1_mes, und Y2_mes zu zwei farbigen Instanzen zusammengefasst. Dabei steht die rote Farbe für X, blau für Y1 und gelb für Y2. Die Instanzen LI und RE senden nur dem roten Anteil, HO und RU den blauen und gelben Anteilen ihre Nachrichten. Der ST-Parameter gilt für alle Richtungen. Die Instanz Stoer erhält ihre

Nachricht nur, wenn alle drei gefärbten Instanzen RndNorm, RndAmp, Rndzeit ihre Information gesendet haben. Die weiterhin von dieser Instanz verschickte Nachricht enthält wiederum auch Information von allen drei Ausgabeinstanzen RndNorm, RndAmp und Rndzeit. Die fünf Parameter für die Bewegung werden auch zu einer gefärbten Instanz zusammengefasst. Dabei lassen sich auch die Nachrichten N1 und N2 zusammenlegen.

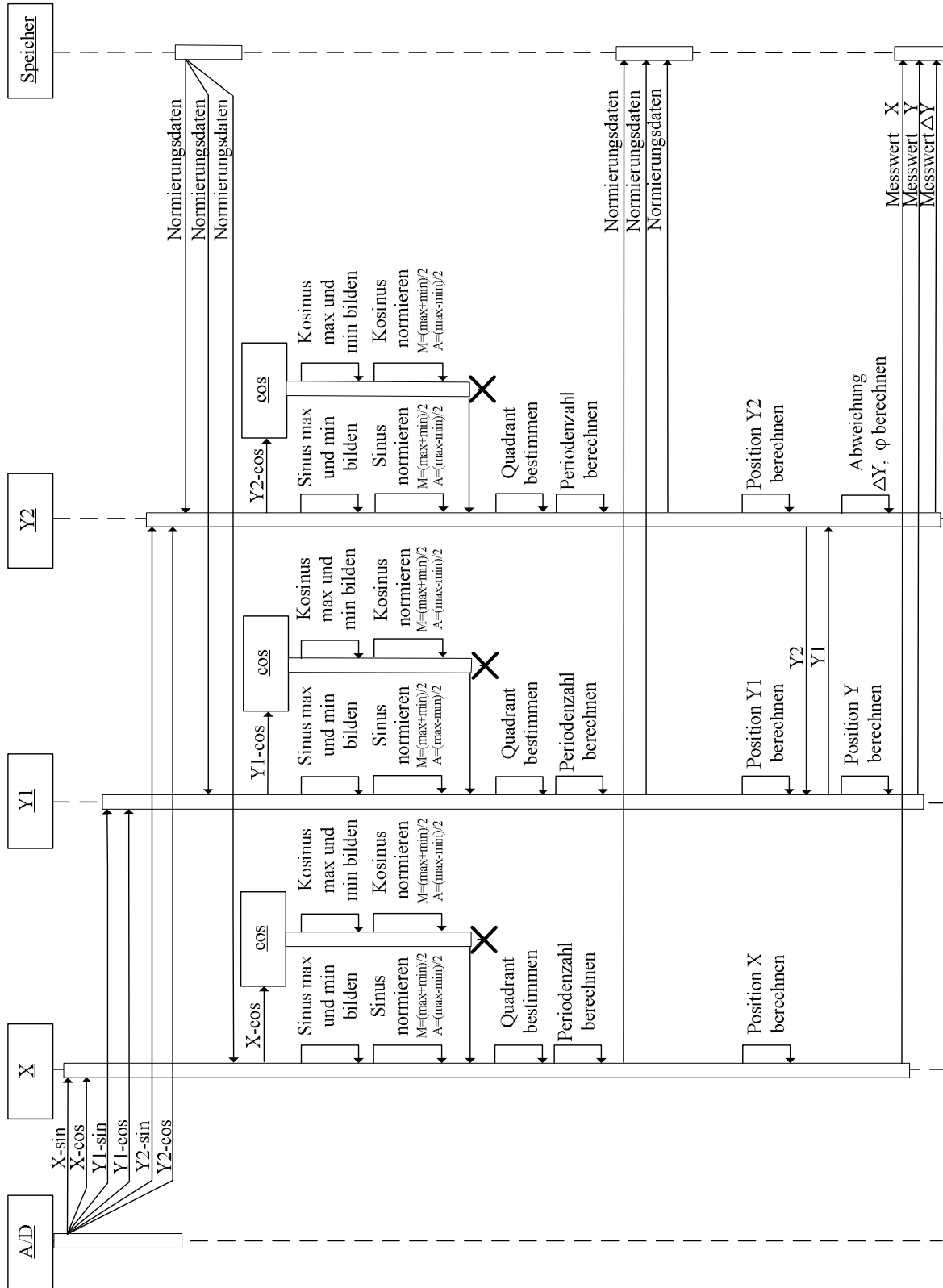


Abbildung 75: Sequenzdiagramm nach UML zur Modellierung des Messsystems

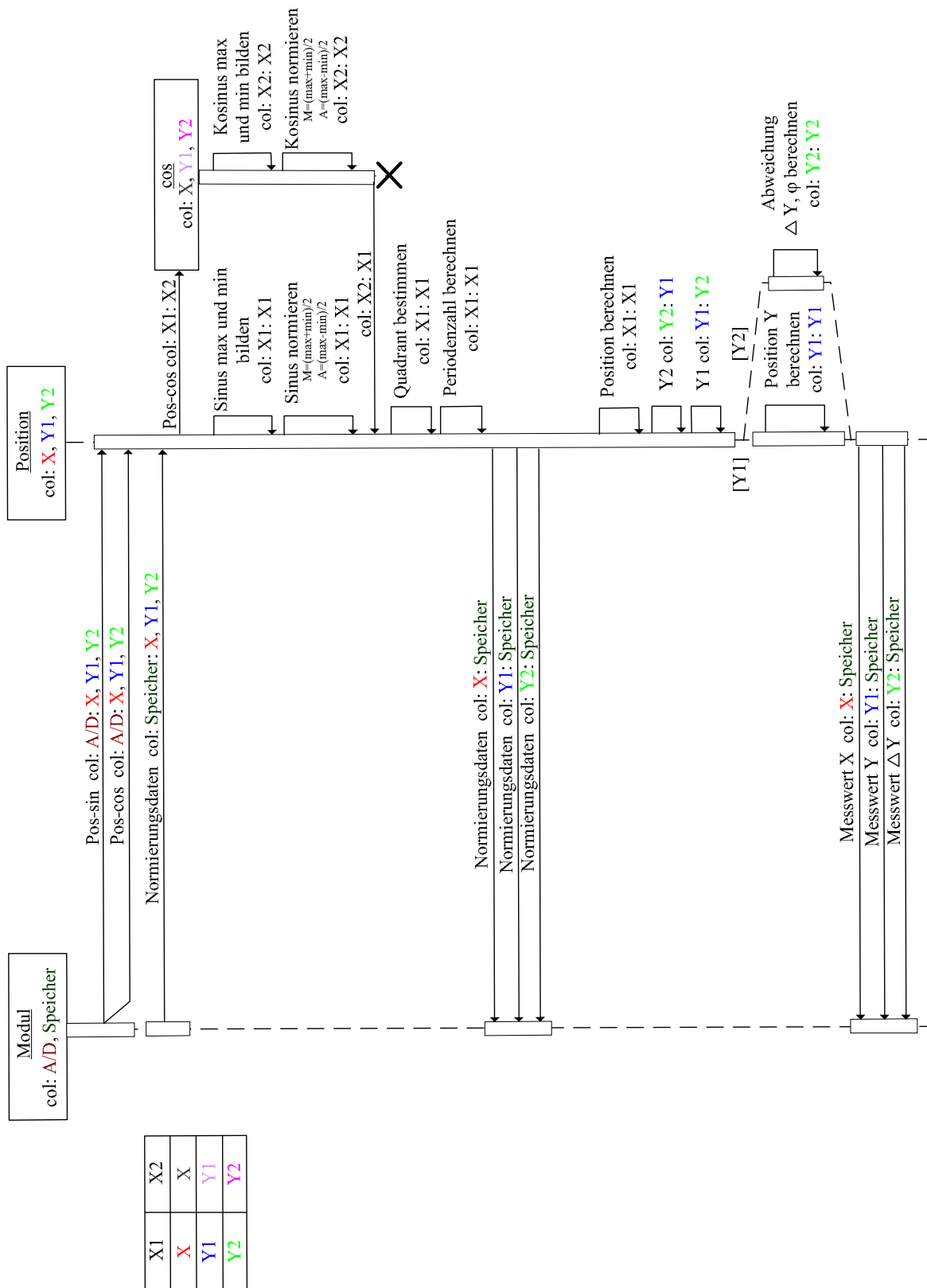


Abbildung 76: Gefärbtes Sequenzdiagramm zur Modellierung des Messsystems

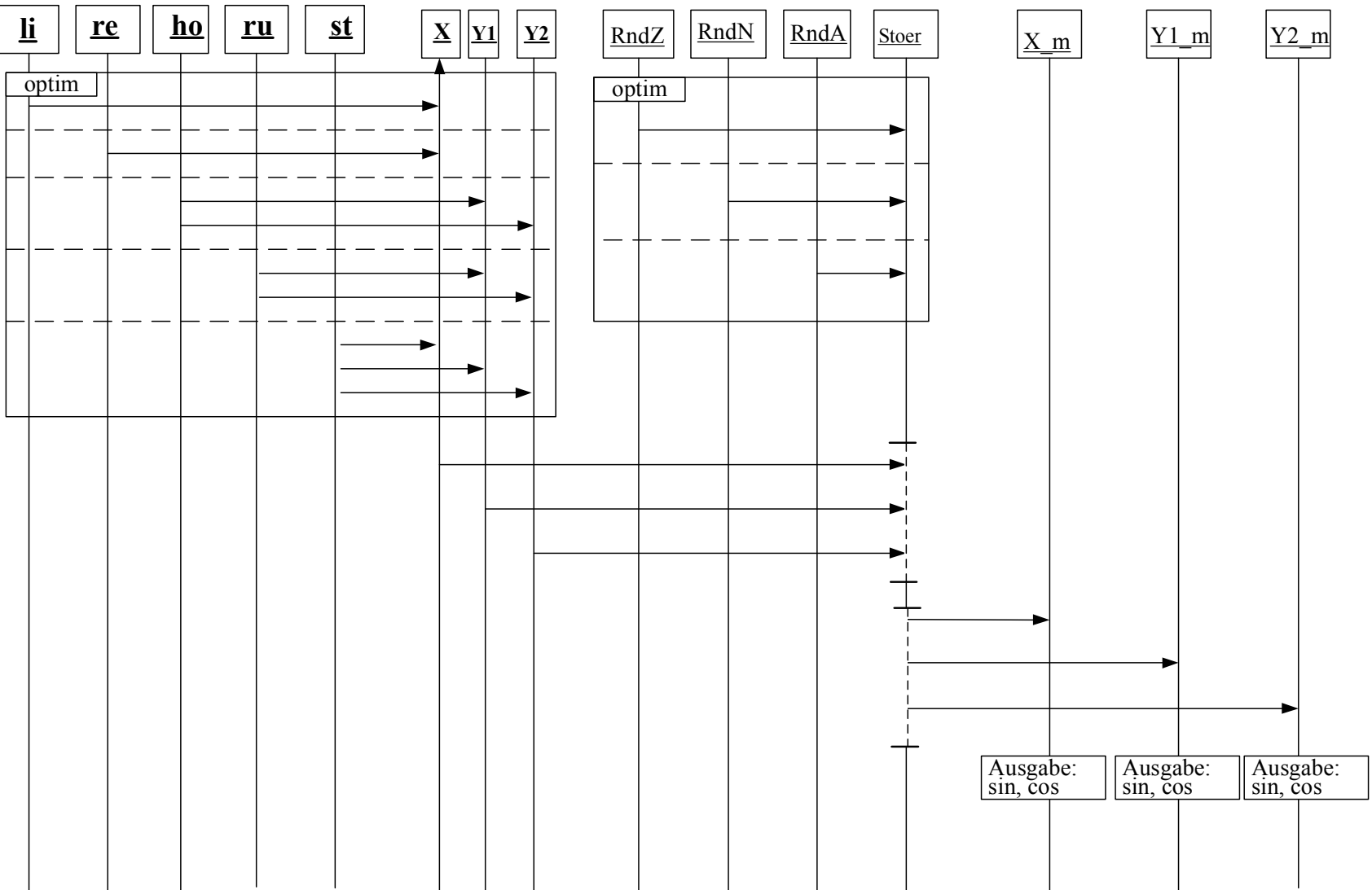


Abbildung 77: Sequenzdiagramm nach UML zur Modellierung der Bewegungen des Mess-
tischs

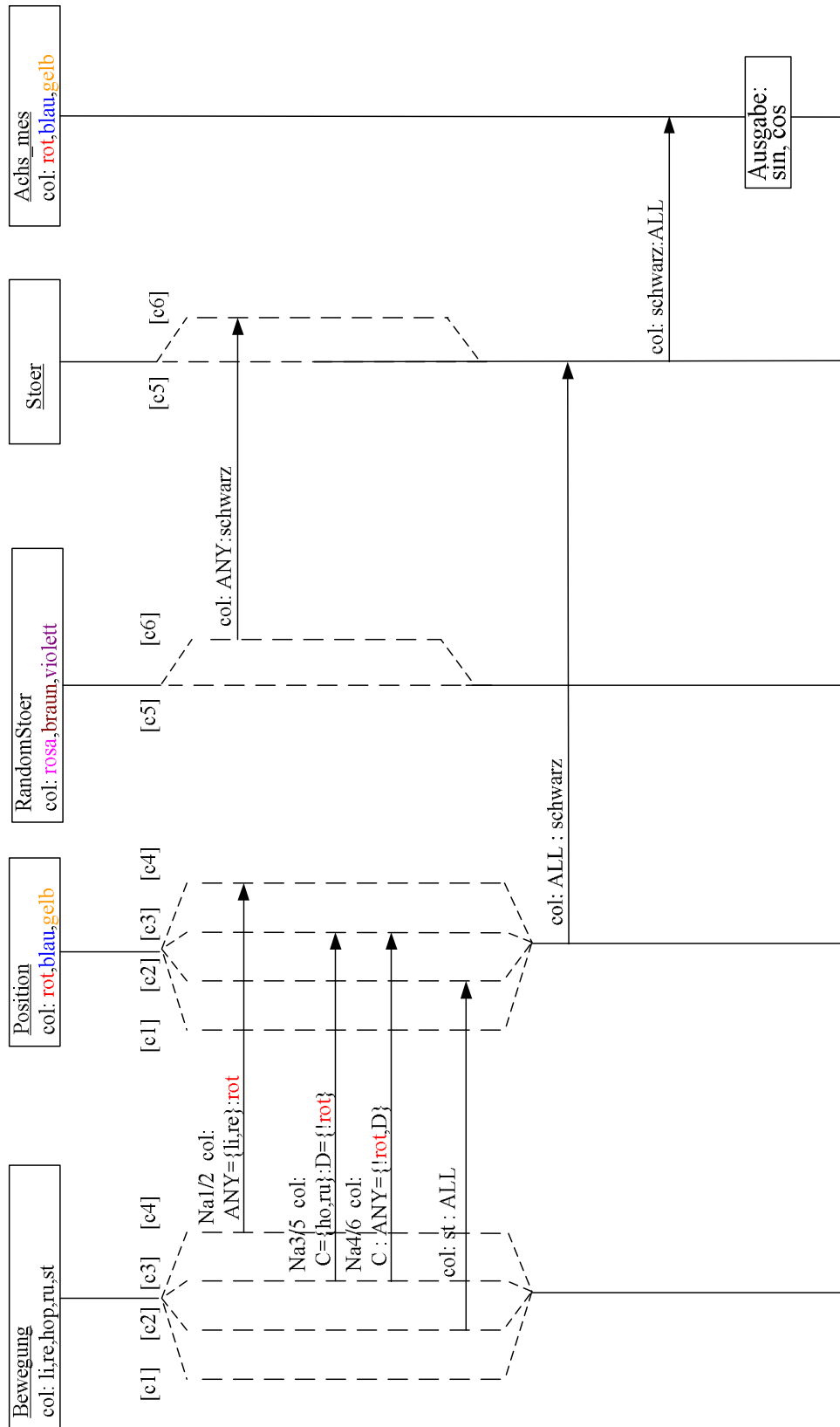


Abbildung 78: Gefärbtes Sequenzdiagramm zur Modellierung der Bewegungen des Messtischs

Die Nachricht Na1/2 enthält jetzt die gleiche Funktion wie in normalen Sequenzdiagrammen die Nachrichten Na1 und Na2 hatten. Sie wird nur für LI oder RE ausgelöst. Gleiches passiert

mit den Nachrichten Na3/5 und Na4/6. Um die richtigen Funktionsweisen beizubehalten, werden dabei die vordefinierte Variable ANY und die neu angelegte Variable C angesetzt. Alle drei optionalen Störungsnachrichten werden genauso zu einer gefärbten Nachricht vereint.

Das entstandene gefärbte Sequenzdiagramm bietet einen besseren Überblick über das Zusammenwirken des Gesamtsystems.

Transformationen zwischen gefärbten Diagrammen über höhere Petri-Netze

In der UML wird die Anwendung der betrachteten Diagrammtypen (Zustandsdiagramme, Aktivitätsdiagramme und Sequenzdiagramme) wie bereits beschrieben, folgendermaßen empfohlen [OMG 2004]:

- Zustandsdiagramme beschreiben das Verhalten von Instanzen von Klassen, aber auch von Anwendungsfällen, Aktoren, Subsystemen, Operationen oder Methoden. Sie werden empfohlen, wenn asynchrone Ereignisse existieren.
- Aktivitätsdiagramme werden als ein Spezialfall der Zustandsdiagramme bezeichnet. Sie können damit prinzipiell für die bei den Zustandsdiagrammen benannten Probleme genutzt werden. Sie werden empfohlen, wenn die meisten Ereignisse aus dem Abschluss interner Aktionen resultieren.
- Das Sequenzdiagramm soll für die Beschreibung des Nachrichtenaustausches zwischen Instanzen in der zeitlichen Abfolge dienen.

Sequenzdiagramme können damit das Kooperationsverhalten von Instanzen von Klassen bzw. innerhalb von Anwendungsfällen, Subsystemen und bei gegenseitigen Methodenaufrufen modellieren. Wenn man den Teil einer Lebenslinie im Sequenzdiagramm zwischen Nachrichtenaustauschpunkten als Zustände auffasst, können alle drei Diagramme mit der gleichen prinzipiellen Methode von Zuständen und Zustandsübergängen mit sequenziellen und parallelen Verzweigungen verschiedene Teile bzw. Abstraktionsebenen eines Systems beschreiben. Dabei ist der Diagrammtyp auch von der Vorliebe des Entwicklers und den betrachteten Aspekten abhängig. Durch die im Weiteren entwickelte Methode der Konvertierung der Diagramme ineinander über höhere Petri-Netze als Zwischennotation werden damit die unterschiedlichen Diagrammtypen kombinierbar. Damit ergibt sich ein gemeinsames Systemmodell, wenn mit unterschiedlichen Diagrammen gearbeitet wurde. Das ermöglicht die Validierung und Verifikation des gesamten Systems bzw. von Subsystemen in deren Umfeld.

In diesem Kapitel werden Transformationsverfahren zwischen den drei bereits beschriebenen gefärbten Aktivitäts-, Zustands- und Sequenzdiagrammen für jede Kombination von Quell-

und Zieldiagramm entwickelt. Dabei werden als Zwischennotation prinzipiell Gefärbte Petri-Netze (CPN) eingesetzt. Das hat den Vorteil, dass für das Quelldiagramm nur eine Teiltransformation in das CPN durchgeführt wird, aus dem dann durch weitere unterschiedliche Teiltransformationen die Zieldiagramme erzeugt werden. Zusätzlich entsteht eine mit den Mitteln der Petri-Netz-Analysetechnik verifizierbare Zwischennotation, so dass die Verifikationssagen auf die Quelldiagramme übertragen werden können.

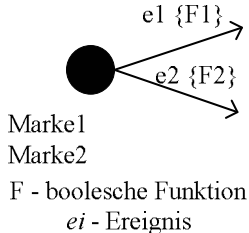
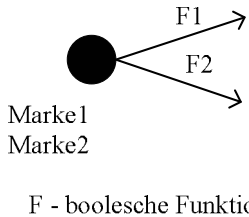
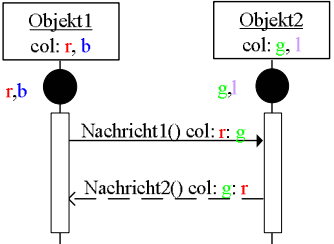
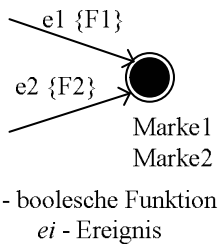
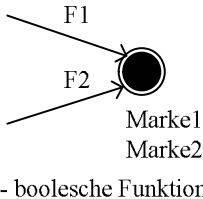
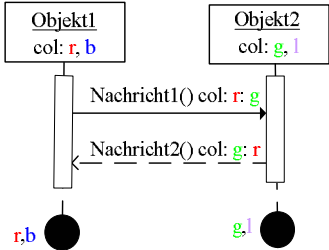






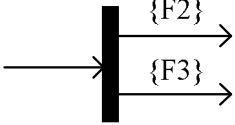
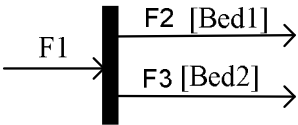
Am Anfang dieses Kapitels wird ein Vergleich von Zustandskonzept, Ereigniskonzept, Zustandsübergangskonstruktionen sowie von verschiedenen Entscheidungs-, Synchronisations- und Vereinigungskonstrukten für gefärbte Zustands-, Aktivitäts- und Sequenzdiagramme durchgeführt. Durch diesen Vergleich wird die Möglichkeit dieses Transformationsvorhabens nachgewiesen. Weiterhin folgt die Beschreibung von Schritten bei der Transformation aller gefärbten Diagramme in Gefärbte Petri-Netze und Rücktransformationen von aus Sequenzdiagrammen entstandenen Gefärbten eingeschränkten Petri-Netzen in gefärbte Zustands- und Aktivitätsdiagramme.

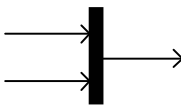
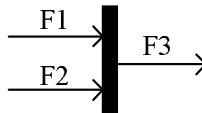
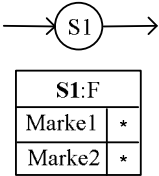
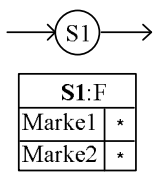
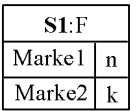
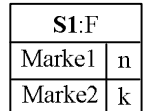
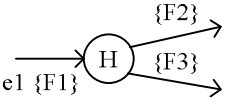
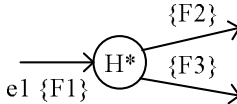
5.1 Elementevergleich von gefärbten Aktivitäts-, Zustands- und Sequenzdiagrammen

Viele Konstrukte und Elemente, sowohl der normalen Diagramme der UML als auch der gefärbten Zustands-, Aktivitäts- und Sequenzdiagramme, haben vergleichbare Funktionen. Die folgende Tabelle 5 (überarbeitet nach [Baru 2002]) zeigt eine Gegenüberstellung von allen wesentlichen Elementen dieser drei Diagrammtypen.

Tabelle 5: Elementvergleich

Bezeichnung des Elementes	gZD	gAD	gSD
	Bezeichnung des Konzepts		
	Zustandskonzept		
Zustand			
Einfacher gefärbter Zustand			
Gefärbter zusammengesetzter Zustand			

<p>Gefärbter Startzustand</p>	 <p>Marke1 Marke2 F - boolesche Funktion <i>ei</i> - Ereignis</p>	 <p>Marke1 Marke2 F - boolesche Funktion</p>	
<p>Gefärbter Endzustand</p>	 <p>Marke1 Marke2 F - boolesche Funktion <i>ei</i> - Ereignis</p>	 <p>Marke1 Marke2 F - boolesche Funktion</p>	
<p>Statische gefärbte Verbindungsstelle</p>			<p>-</p>
<p>Dynamische gefärbte Entscheidungsstelle</p>			<p>-</p>
<p>Gefärbter Entscheidungsknoten</p>			<p>-</p>
<p>Synchronisation (gefärbte Gabelung)</p>	 <p>Ereignis {Empfänger} {F} - boolesche Funktion</p>	 <p>F - boolesche Funktion</p>	<p>-</p>

<p>Synchronisation (gefärbte Vereinigung)</p>	 <p>Ereignis {Empfänger}</p>	 <p>F - boolesche Funktion</p>	-
<p>Unbeschränkter gefärbter Synch-Zustand</p>	 <p>F - boolesche Funktion</p>	 <p>F - boolesche Funktion</p>	-
<p>Beschränkter gefärbter Synch-Zustand</p>	 <p>F - boolesche Funktion</p>	 <p>F - boolesche Funktion</p>	-
<p>Gefärbter History-Zustand</p>	 <p>F - boolesche Funktion <i>ei</i> - Ereignis</p>	-	-
<p>Gefärbter tiefer History-Zustand</p>	 <p>F - boolesche Funktion <i>ei</i> - Ereignis</p>	-	-

<p>Ereignis</p>			
<p>Transitionskonzept</p>			
<p>Transition</p>			
<p>Einfache gefärbte Transition</p>			
<p>Komplexe gefärbte Transition</p>		<p style="text-align: center;">-</p>	<p style="text-align: center;">-</p>

<p>Segmentierte gefärbte Transition</p>	<p>ei - Ereign, [Bi] - ÜberBed, {F} - bool Funkt</p>	<p>-</p>	<p>-</p>
<p>Parallelitätskonzept</p>			
<p>Einfache Parallelität</p>	<p>UND-Verfeinerung</p> <p>Ereignis1 {Empfänger} Ereignis2 {Empfänger}</p>		
<p>Parallelität der Verarbeitung der gefärbten Marken</p>	<p>{F} - boolesche Funktion</p>		

		Entscheidungskonzept		
<p>Entscheidung</p>				
		Vereinigungskonzept		
<p>Vereinigung</p>				
		Synchronisationskonzept		
<p>Synchronisation</p>				

5.2 Transformation von gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen in höhere Petri-Netze

Die Transformationen für alle drei Diagrammtypen können an Hand von folgendem Grundalgorithmus durchgeführt werden:

- Aufspaltung des gefärbten Diagramms in Bausteine. In Abbildung 79 ist ein Beispiel für die Aufspaltung eines Sequenzdiagramms gezeigt. Dabei wird die Lebenslinie jedes gefärbten Objekts des gefärbten Sequenzdiagramms in Abschnitte eingeteilt, die jeweils durch eine Nachricht voneinander abgegrenzt werden können. Alle Abschnitte der Lebenslinien der gefärbten Objekte, die mit der entsprechenden Nachricht verbunden sind, bilden einen so genannten Baustein (Abbildung 79 nach [Baru 2002]).

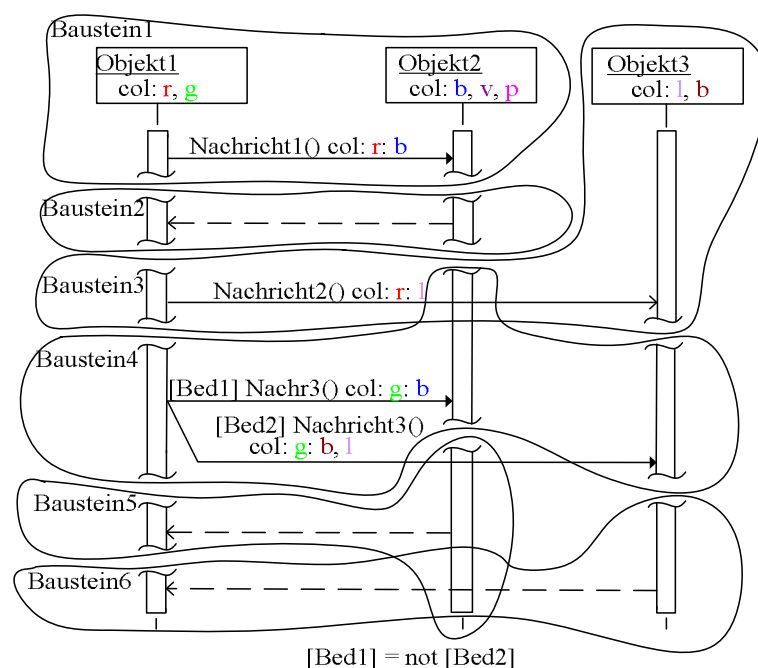


Abbildung 79: Lebenslinienaufspaltung im gefärbten Sequenzdiagramm

- Transformation aus den gefärbten Diagrammbausteinen in Gefärbte Petri-Netz-Bausteine
- Vereinigung oder Verschmelzung der entstandenen Petri-Netz-Bausteine zu einem Gefärbten Petri-Netz. In diesem Schritt werden die entstandenen Bausteine des Gefärbten Petri-Netzes, speziell deren äußere gefärbte Begrenzungsplätze, miteinander verschmolzen. Dabei entsteht das vollständige Gefärbte Petri-Netz, das dem gefärbten Zustands-, Aktivitäts- oder Sequenzdiagramm entspricht.

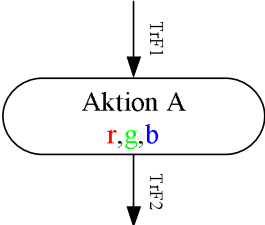
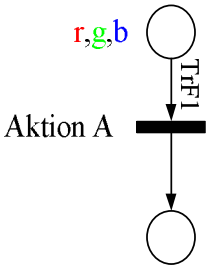
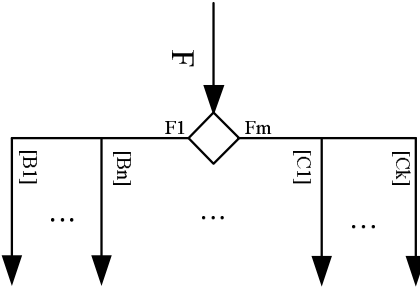
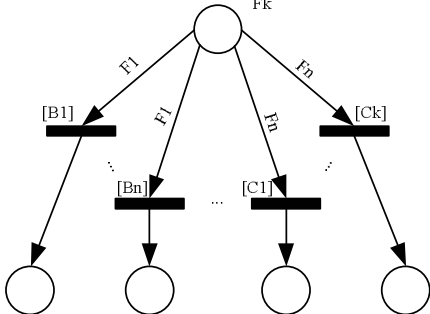
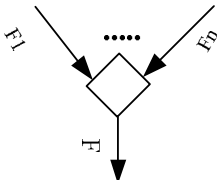
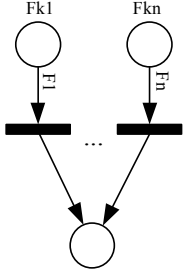
Basierend auf schon bekannten Transformationsverfahren für die normalen Diagramme der UML [Mühl 98, Fric 96, Rang 2002] wird hier in den folgenden Tabellen ein für die gefärbten Diagramme entwickeltes Transformationsverfahren von deren einzelnen Elementen in Elemente Gefärbter Petri-Netze gezeigt.

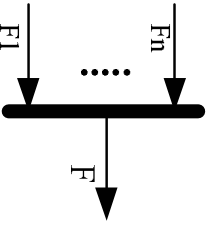
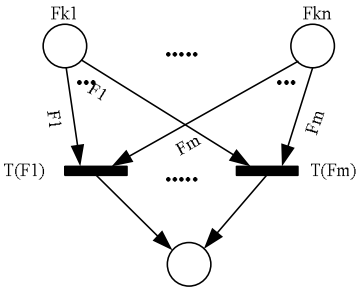
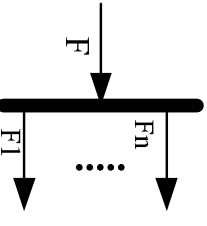
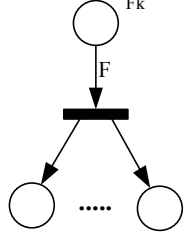
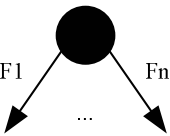

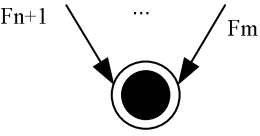

Als erstes wird die Transformation von Elementen des gefärbten Aktivitätsdiagramms in Elemente des Gefärbten Petri-Netzes dargestellt (Tabelle 6 nach [Riab 2001]). Es muss dabei

berücksichtigt werden, dass das bei dieser Transformation entstehende Petri-Netz folgende Eigenschaften aufweist:

- Für eine Farbe ist die Platzkapazität maximal gleich 1,
- die Vielfachheit für eine Farbe kann nicht größer als 1 sein,
- bei Transitionsübergängen passiert kein Farbänderungsprozess: dabei erzeugen alle Transitionen immer Marken der gleichen Farbe in den Nachbedingungen, die sie beim Schalten in den Vorbedingungen abziehen. Dabei hängt die Anzahl der produzierten und konsumierten Marken von den entsprechenden Vor- und Nachkanten ab.
- Die von einer Transition wegführenden Kanten werden mit einer Funktion versehen, die alle Farben von den hinführenden Kanten enthält.

Tabelle 6: Transformation von Elementen des gefärbten Aktivitätsdiagramms in Elemente des Gefärbten Petri-Netzes

Element	Gefärbter Aktivitätsdiagramm-Baustein	Gefärbter Petri-Netz-Baustein
Gefärbte Aktion		
Entscheidung		
Vereinigung		

Synchronisation		
Verzweigung		
Startzustand		
Endzustand		

In der nachfolgenden Tabelle 7 nach [Slav 2001] ist die Transformation von Elementen gefärbter Zustandsdiagramme in Elemente Gefärbter Petri-Netze dargestellt, dabei gehört zu den Schwerpunkten der Transformation folgendes:

- Transformation von Platzkapazitäten,
- Bestimmung der Transitionsschaltmodi,
- Transformation von hierarchischen Strukturen,
- Transformation von Zeitereignissen und zeitbewerteten Transitionen.

Bei den Transformationen von gefärbten Zustandsübergängen in Gefärbte Petri-Netze muss man folgendes berücksichtigen:

- Konsumierte und produzierte Farbmengen bleiben immer gleich.
- Die Anzahl der produzierten Marken wird durch die Kantenfunktionen geregelt.

- Transformation von farbabhängigen Zeitbewertungen.
- Die Schaltungsmodi sind von der entsprechenden Transition des Zustandsdiagramms abhängig.

Die Eigenschaften des bei der Transformation entstandenen Gefärbten Petri-Netzes sind:

- Die Platzkapazität für jede Farbe kann nur 1 sein.
- Die Vielfachheit für eine Farbe kann nur 1 sein,
- Das Schalten von Transitionen verändert die Farbmengen nicht.

Tabelle 7: Transformation von Elementen gefärbter Zustandsdiagramme in Elemente Gefärbter Petri-Netze

Element	Gefärbter Zustandsdiagrammbaustein	Gefärbter Petri-Netz-Baustein
Zustand		
Transition		
Gabelung		
Vereinigung		
Transformation von Hierarchie		

Zeitereignisse		
Startzustand		
Endzustand		

Bei der Umwandlung von gefärbten Sequenzdiagrammen in Gefärbte Petri-Netze wird eine Instanz, wie in der nachfolgenden Tabelle 8 nach [Rang 2001] und [Baru 2002] gezeigt ist, durch ein lineares Petri-Netz, welche eine Folge von durch Transitionen verbundenen Plätzen darstellt, ersetzt. Die weiter oben eingeführten Zustände von Instanzen in Sequenzdiagrammen werden in Petri-Netzen durch Plätze repräsentiert. Der erste Platz von diesem Petri-Netz enthält einzelne Marken in verschiedenen Farben, welche entsprechend in der gefärbten Instanz vorhanden sind. Dabei sind die Platzkapazitäten mit 1 für jede Farbe vorzusehen. Das Sequenzdiagramm wird folglich in ein Petri-Netz überführt, welche mehrere solche Teilnetze enthält.

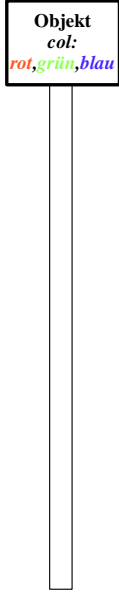
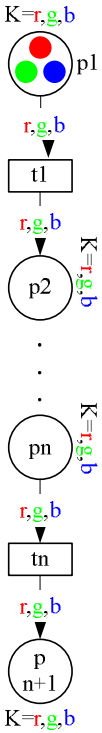
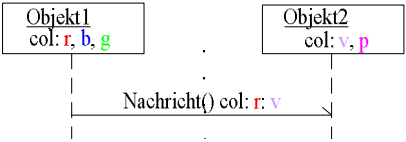
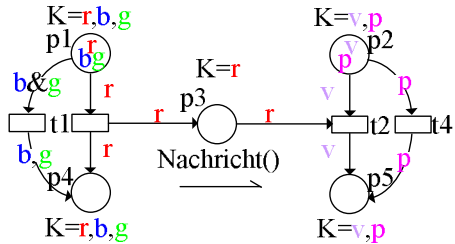
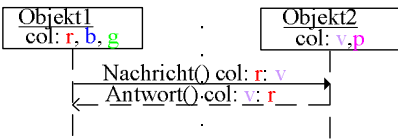
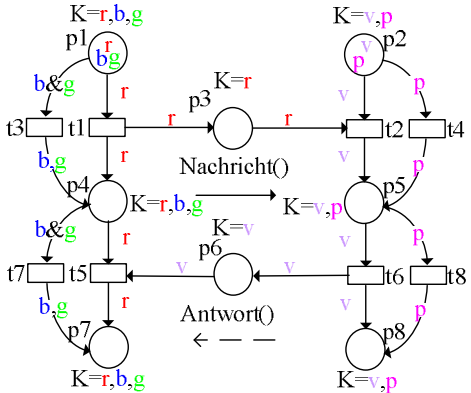
Die Substitution von Nachrichten hängt von ihrem Typ ab. Abhängig von der Farbkonvertierungsfunktion werden folgende Fälle bei der Umwandlung in Petri-Netze unterschieden:

- Die Konvertierungsfunktion besitzt Farbkonstanten (col: const), das Ereignis findet in diesem Fall nur für bestimmte Farben statt. Deshalb sind beim Versenden oder Empfangen der Nachricht nur diese Farben beteiligt und alle anderen werden durch zusätzliche Transitionen behandelt, die mit keinen Nachrichten verbunden sind.
- Die Konvertierungsfunktion besitzt die vordefinierte Farbvariable ANY (col: ANY), welche definiert, dass nur die erste Farbe für das Verschicken oder Empfangen der Nachricht zuständig ist. Zum Modellieren dieser Situation wird ein zusätzlicher Platz mit Testkante benötigt.
- Die Konvertierungsfunktion besitzt die vordefinierte Farbvariable ALL (col: ALL), dabei wird ein zusätzlicher Platz im Petri-Netz benötigt, um alle Farben zu sammeln und danach eine Nachricht für alle Farben zu verschicken oder zu empfangen.

Beispiele zur Behandlung von verschiedenen Nachrichtentypen sind in der nachfolgenden Tabelle 8 gezeigt.

Es gibt auch komplexere Bausteine der gefärbten Sequenzdiagramme, die in die entsprechenden Bausteine der Gefärbten Petri-Netze umgeformt werden müssen. Beispielsweise entstehen bei Parallelität und Entscheidungskonstruktionen die in nachfolgender Tabelle gezeigten CPN.

Tabelle 8: Transformation von Elementen gefärbter Sequenzdiagramme in Elemente Gefärbter Petri-Netze

Element	Gefärbter Sequenzdiagramm- baustein	Gefärbter Petri-Netz-Baustein
Instanz		
Asynchrone Nachricht		
Synchrone Nachricht		

<p>Nachricht mit vor-definierter Farbvariable ANY</p>		
<p>Nachricht mit vor-definierter Farbvariable ALL</p>		
<p>Entscheidungskonstruktion</p>		
<p>Parallele Nachrichtenversendung</p>		

5.3. Eigenschaften von eingeschränkten, durch Transformation entstandenen Petri-Netzen

Bei der in Kapitel 5.2. beschriebenen Transformation aus gefärbten Aktivitäts-, Zustands- und Sequenzdiagrammen entstehen eingeschränkte Gefärbte Petri-Netze. Diese Einschränkungen resultieren aus den Beschreibungsmöglichkeiten der Ausgangsdiagramme, die nicht alle Möglichkeiten der allgemeinen Gefärbten Petri-Netze aufweisen. Bei allen betrachteten Ausgangsdiagrammen sind sie gleich. Das ermöglicht einfachere Transformationsalgorithmen. Die entstandenen Petri-Netze haben folgende Eigenschaften:

- Alle Plätze haben als maximale Kapazität 1 für eine Farbe.
- Alle Kanten haben als maximale Vielfachheit 1 für eine Farbe.
- Für die Aktivitäts- und Zustandsdiagramme gibt es genau einen Platz ohne Vorkanten. Dieser Platz entspricht dem Startzustand und ist unter der Anfangsmarkierung als einziger mit allen Marken des Modells belegt. In Sequenzdiagrammen gibt es mehrere Plätze ohne Vorkanten, die für mehrere Objekte aktuell sind.
- Die endliche Menge von Plätzen ohne Nachkanten entspricht den Endzuständen im gefärbten Zeitintervallzustands-, Aktivitäts-, Sequenzdiagramm.
- Alle Transitionen in Zustands- und Aktivitätsdiagrammen erzeugen immer dieselbe Marke, die sie konsumieren – die Farben ändern sich nicht. Abhängig von den Nachrichtentypen in Sequenzdiagrammen kann es zu Farbänderungen kommen.
- Die Plätze ohne Färbungsfunktionen sind Plätze, deren Färbungsfunktionen der Startplatzfunktion gleich sind.
- Kanten ohne Inzidenzfunktionen sind Kanten, deren Inzidenzfunktionen aus der Disjunktion aller Farben des Modells bestehen.

5.4. Transformation eingeschränkter Petri-Netze in Zustands-, Aktivitäts- und Sequenzdiagramme

Im vorliegenden Kapitel wird ein Algorithmus zur Überführung gefärbter eingeschränkter Petri-Netze nach Kapitel 5.3. in gefärbte Diagramme gezeigt. Dabei ist in den Abbildungen als Beispiel ein bei der Transformation aus einem Sequenzdiagramm entstandenes gefärbtes eingeschränktes Petri-Netz dargestellt. Dieses wird in ein Zustandsdiagramm transformiert. Es werden folgende Schritte bei einer derartigen Transformation benötigt:

▪ Schritt 1: Platznummerierung

Alle Plätze des gefärbten Petri-Netzes müssen eindeutig durchnummeriert werden. Dabei kann auf die explizite Nummerierung der Interaktionsplätze optional verzichtet werden, da es hierfür keine explizite Notwendigkeit gibt.

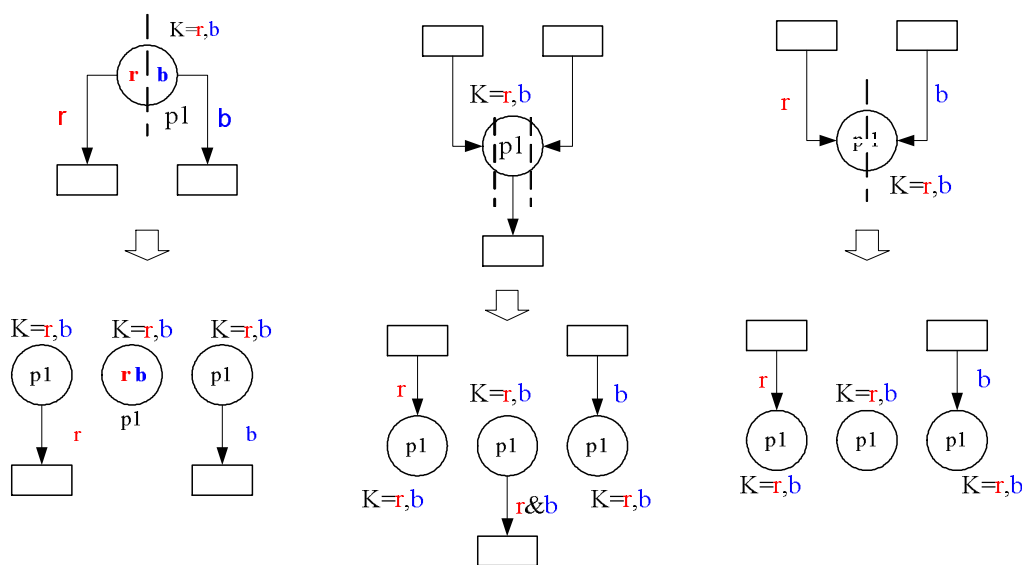


Abbildung 80: Platzaufspaltung und Platzvervielfältigung eines Startplatzes, eines Zwischenplatzes und eines Endplatzes

▪ Schritt 2: Platzaufspaltung und Platzvervielfältigung

Im zweiten Schritt wird jeder Platz entsprechend seiner Vor- bzw. Nachtransitionen aufgespalten und mit seiner Nummerierung vervielfältigt (Abbildung 80 nach [Rang 2001] und [Baru 2002]). Dank diesem Zerlegungsvorgang bekommt man zum Schluss einzelne gefärbte Petri-Netz-Bausteine, welche geeignet zu weiteren Transformationen in die gefärbten Diagramme sind.

▪ Schritt 3: Baueintransformation

In diesem Schritt wird jeder entstandene Baustein des gefärbten Petri-Netzes durch einen entsprechenden Baustein des gefärbten Zustands-, Aktivitäts- oder Sequenzdiagramms ersetzt. In den Abbildungsbeispielen werden entsprechende Plätze vom gefärbten Petri-Netz in die gefärbten Objektzustände des gefärbten Zustandsdiagramms transformiert. Dabei entstehen anonyme Zustände. Einige Bausteine der gefärbten Petri-Netze, wie Startplatz, Endplatz, Zustandswechsel und Pseudozustandswechsel, welche zum Beispiel bei Iterationen entstehen

können, werden spezifisch in Bausteine der Zustandsdiagramme umgewandelt. In Abbildung 81 nach [Baru 2002] sind diese verschiedenen Zustandsdiagramm-Bausteine gezeigt.

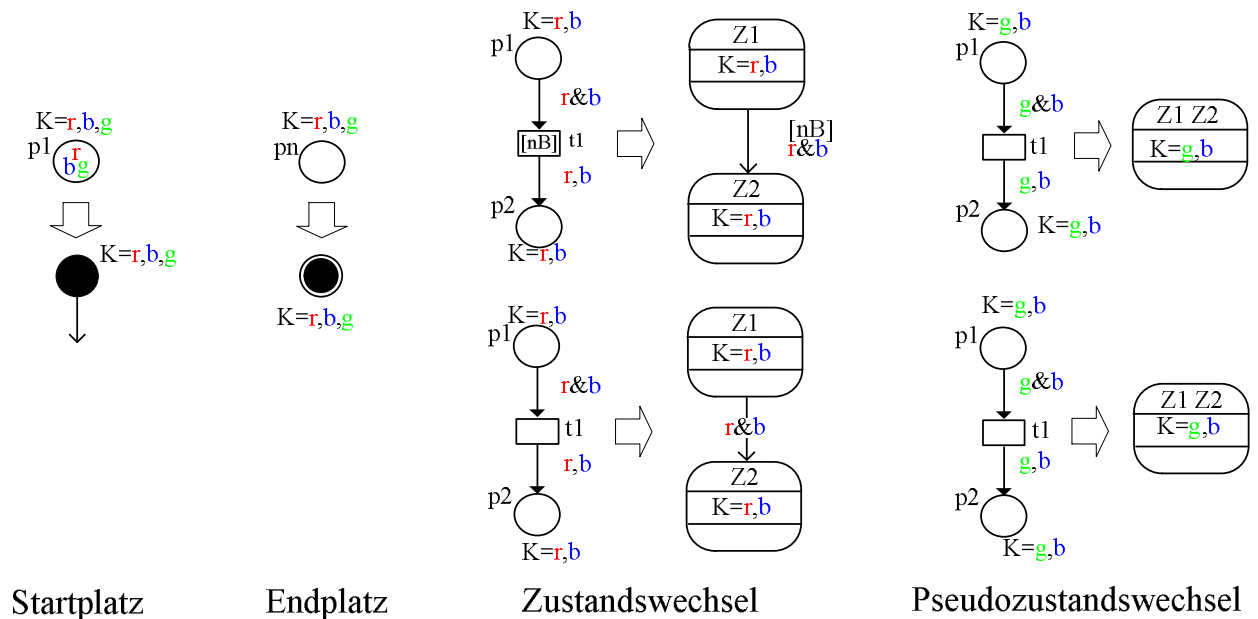


Abbildung 81: Transformationsschema für einen Startplatz, Endplatz, Zustandswechsel und Pseudozustandswechsel aus dem Gefärbten Petri-Netz

In Abbildung 82 nach [Baru 2002] ist die Transformation von Nachrichten mit Konvertierungsfunktionen, die Farbkonstanten und vordefinierte Farbvariablen vom Typ ANY besitzen, gezeigt. In dem Fall, wenn die Farbkonvertierungsfunktion Farbkonstanten enthält, wird jeder Platz, ausgenommen ein Interaktionsplatz, vom Gefärbten Petri-Netz in einen gefärbten Zustand umgewandelt. Kapazitäten bleiben dabei erhalten. Das sendende Objekt wird durch ein Ereignis „/Send“ gesteuert. In Konstrukten, in denen die Farbkonvertierungsfunktion die vordefinierte Variable ANY erhält, werden Hilfsplätze (Plätze P6 und P7, siehe Abbildung 82) zur Ableitung von nicht bei der Versendung von Nachrichten beteiligten Objekten nicht durch gefärbte Zustände repräsentiert. Dieses Konstrukt wird durch die Zuordnung von booleschen Funktionen zu Transitionen modelliert.

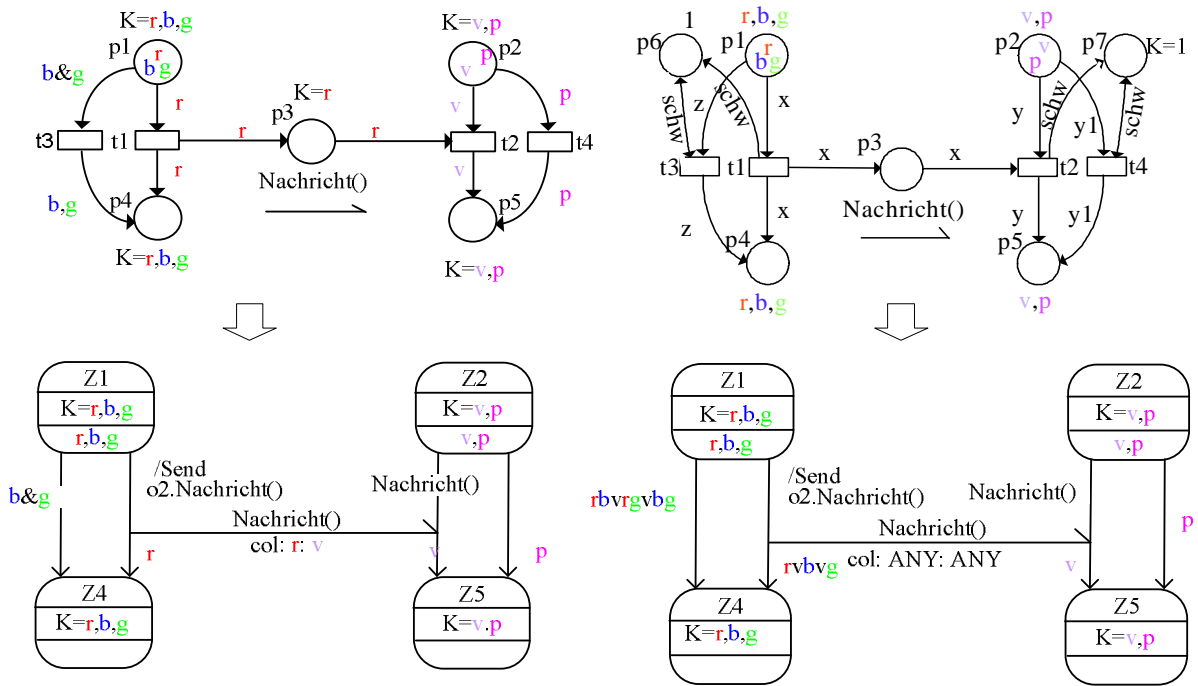


Abbildung 82: Transformationsschema für Nachrichten mit Konvertierungsfunktionen, die Farbkonstanten und vordefinierte Farbvariablen ANY enthalten

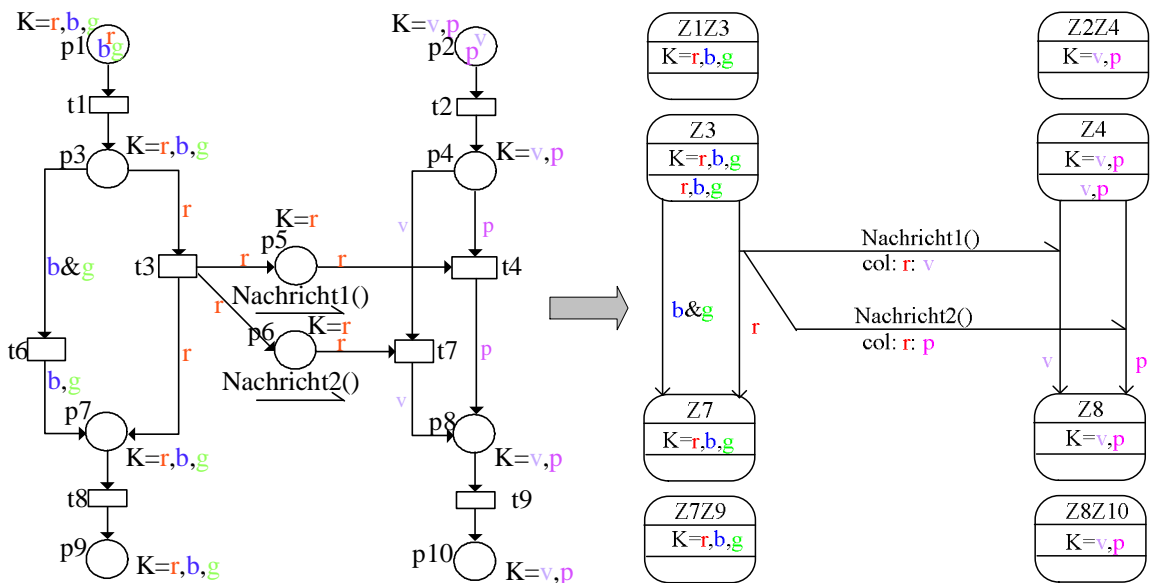


Abbildung 83: Transformationsschema für die Bausteine des Gefärbten Petri-Netzes, die der Parallelität entsprechen

Eine Besonderheit der Realisierung der Parallelität mit den Mitteln der gefärbten Zustandsdiagramme besteht darin, dass bei der Transformation eines Bausteins des Gefärbten Petri-Netzes in den entsprechenden Baustein des gefärbten Zustandsdiagramms so genannte gefärbte Pseudozustände entstehen (Abbildung 83). Diese Zustände entstehen bei der Transformation der ihnen entsprechenden Pseudozustandswechsel des Gefärbten Petri-Netzes. Dabei stellt

ein solcher Zustand eigentlich nur ein Bindeglied dar und wird somit mit den regulären gefärbten Vor- und Nachzuständen zu einem gefärbten Zustand verschmolzen.

In Abbildung 84 nach [Baru 2002] ist die Transformation eines Entscheidungselements in ein gefärbtes Zustandsdiagramm dargestellt. Bei dieser Transformation werden allen Zustandsübergängen im gefärbten Zustandsdiagramm entsprechende Überwachungsbedingungen zugeordnet, so, dass die Nachricht nur bei der Ausführung dieser Überwachungsbedingung übergeben werden kann.

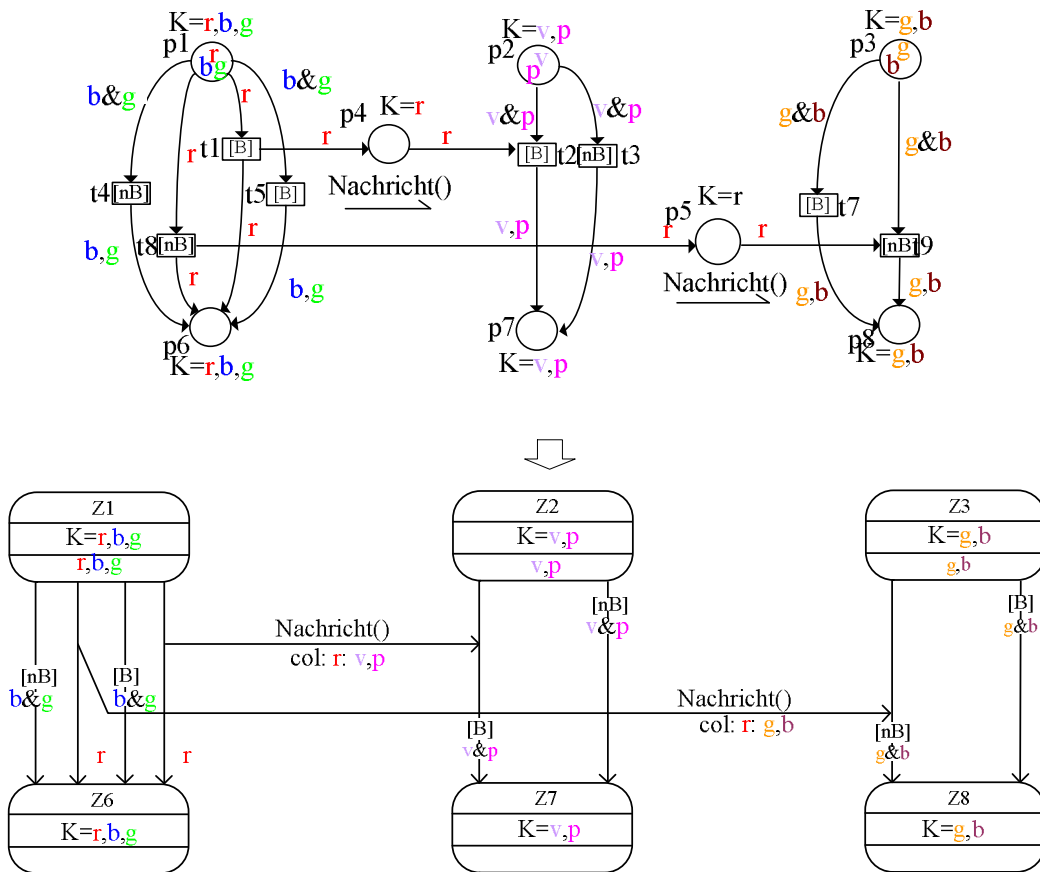


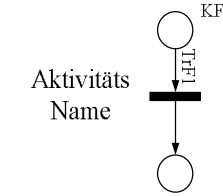

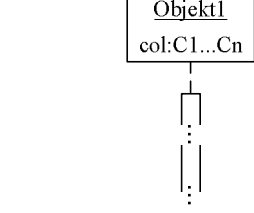
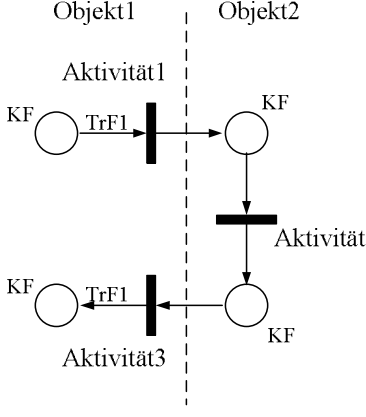
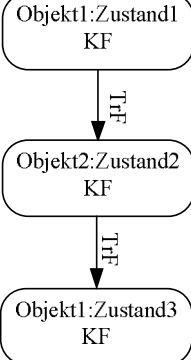
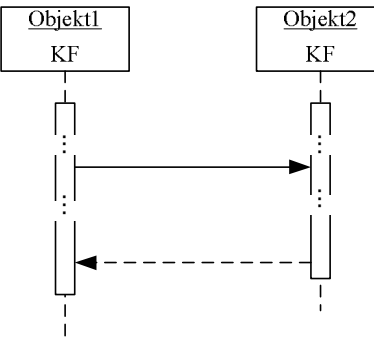
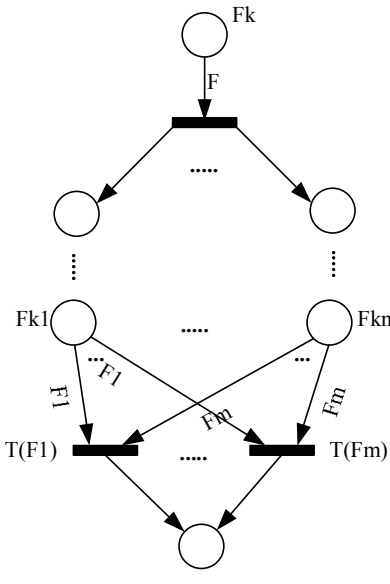
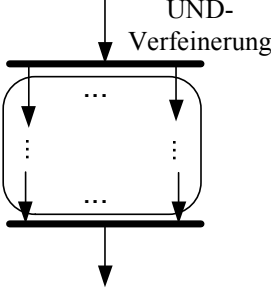
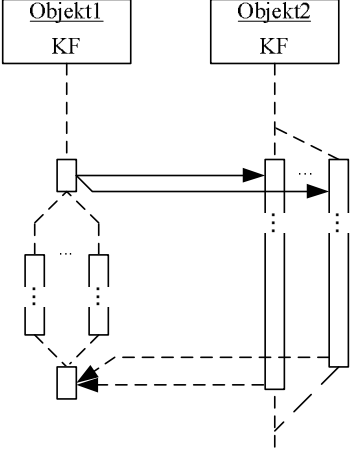
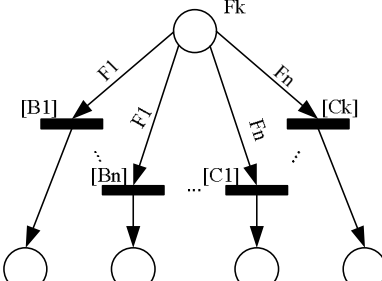
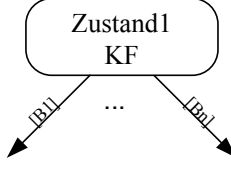
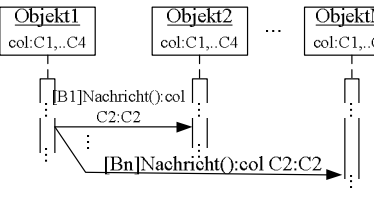
Abbildung 84: Transformationsschema für die Bausteine des Gefärbten Petri-Netzes, die die Entscheidungskonstruktion darstellen

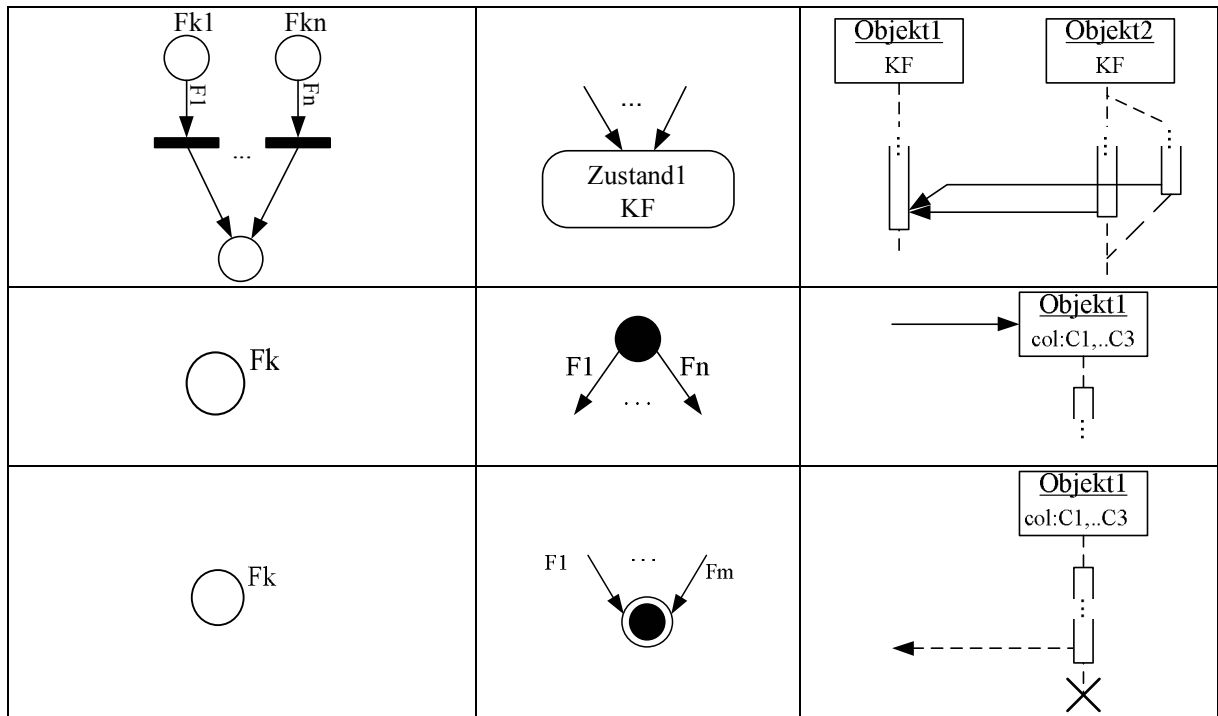
▪ Schritt 4: Bausteinverschmelzung

Die Bausteinverschmelzung stellt den letzten Schritt dieses Transformationsverfahrens dar. Hier werden alle vollständig und teilweise gleichnamigen gefärbten Zustände miteinander verschmolzen. Bei der Verschmelzung eines gefärbten Endzustandes mit einem normalen gefärbten Zustand muss dabei berücksichtigt werden, dass sich beide gefärbte Zustände bezüglich der grafischen Darstellungsform voneinander unterscheiden. Demzufolge wird der normale gefärbte Zustand durch den gleichnamigen gefärbten Endzustand ersetzt.

In nachfolgender Tabelle 9 [Riab 2003] ist ein Transformationsverfahren von wesentlichen, bei der Überführung von gefärbten Aktivitätsdiagrammen in Gefärbte Petri-Netze entstandenen Petri-Netz-Bausteinen in Elemente der gefärbten Zustands- und Sequenzdiagramme dargestellt.

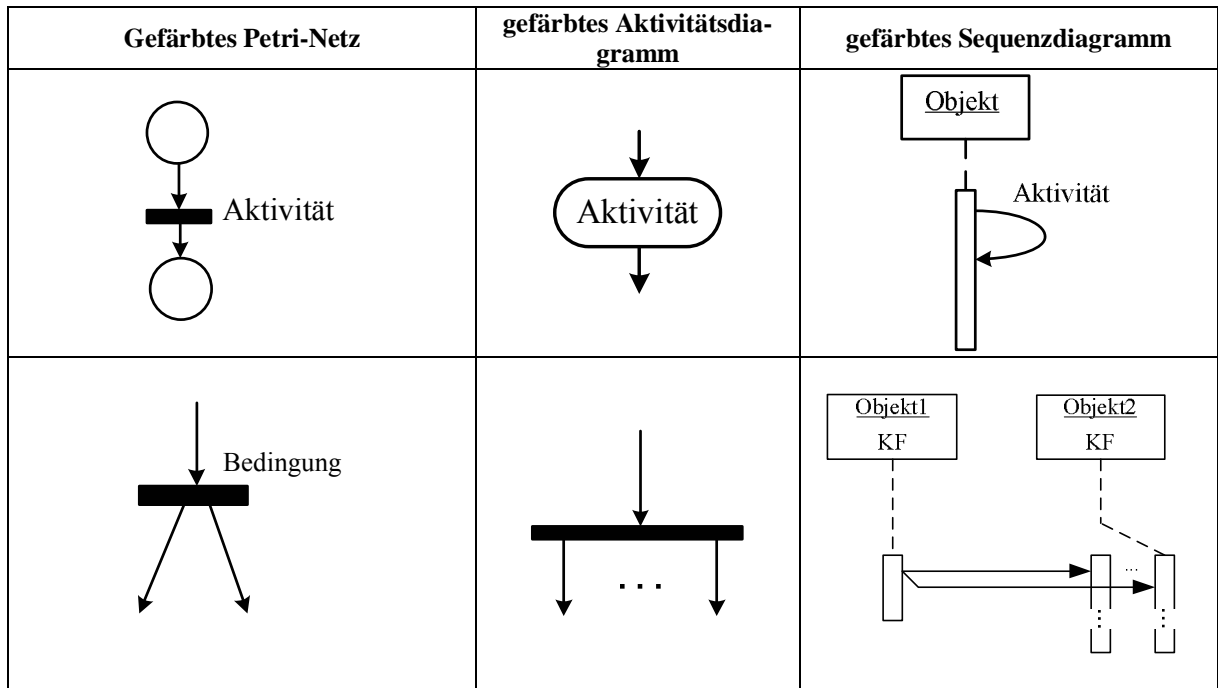
Tabelle 9: Transformation Gefärbter Petri-Netze in Bausteine gefärbter Zustands- und Sequenzdiagramme

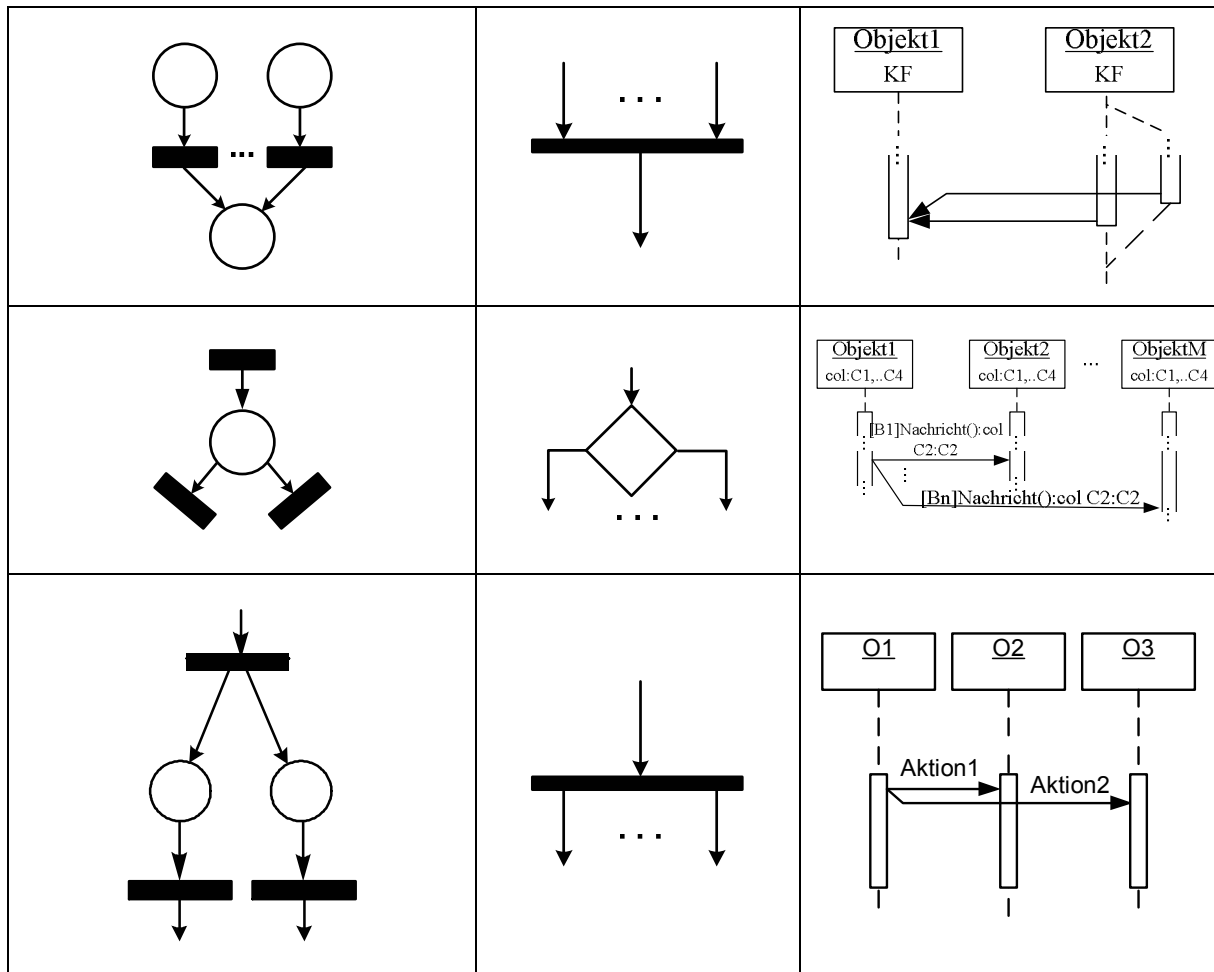
Gefärbtes Petri-Netz	gefärbtes Zustandsdiagramm	gefärbtes Sequenzdiagramm
		
		
		
		



In der nachfolgenden Tabelle 10 ist die Umwandlung von aus gefärbten Zustandsdiagrammen entstandenen Gefärbten Petri-Netzen in gefärbten Aktivitäts- und Sequenzdiagramme dargestellt.

Tabelle 10: Transformation Gefärbter Petri-Netze in Bausteine gefärbter Aktivitäts- und Sequenzdiagramme





5.5 Transformation von Zustands-, Aktivitäts- und Sequenzdiagrammen ineinander über Gefärbte Petri-Netze am Beispiel von einem Messsystems

Das in Kapitel 2 beschriebene und mittels gefärbten Diagrammen in Kapitel 4 modellierte Beispiel eines Messsystems wird hier in Ausschnitten gezeigt. In den Abbildung 85 ist ein gefärbtes Zustandsdiagramm und die Verfeinerung von einem Zustand aus diesem Zustandsdiagramm dargestellt, welche wesentliche Schritte des modellierten Messsystems aufweist.

Dieses gefärbte Zustandsdiagramm lässt sich ziemlich einfach in ein Gefärbtes Petri-Netz überführen (Abbildung 85 [Slav 2003]). Dabei sind die in den Zuständen durchzuführenden Aktivitäten nicht im Petri-Netz gezeigt. Da aber die Information über diese Aktivitäten sehr wichtig für die weiteren Überführungen in andere gefärbte Diagramme sind, müssen diese sowie die Schaltbedingungen und Ereignisse erhalten bleiben, z.B. in der Tabellenform, die unten angegeben ist (Tabelle 11 nach [Slav 2003]).

In den weiteren Abbildungen 87, 88, 89 [Slav 2003] sind die ersten Umwandlungsschritte der Petri-Netz-Bausteine, die bei den Transformationen entstanden sind, in Bausteine von Aktivitäts- und Sequenzdiagrammen gezeigt. In den Abbildungen 90, 91 [Slav 2003] werden die nach der Transformation aus dem Zustandsdiagramm entstandenen gefärbten Sequenz- und Aktivitätsdiagramme gezeigt.

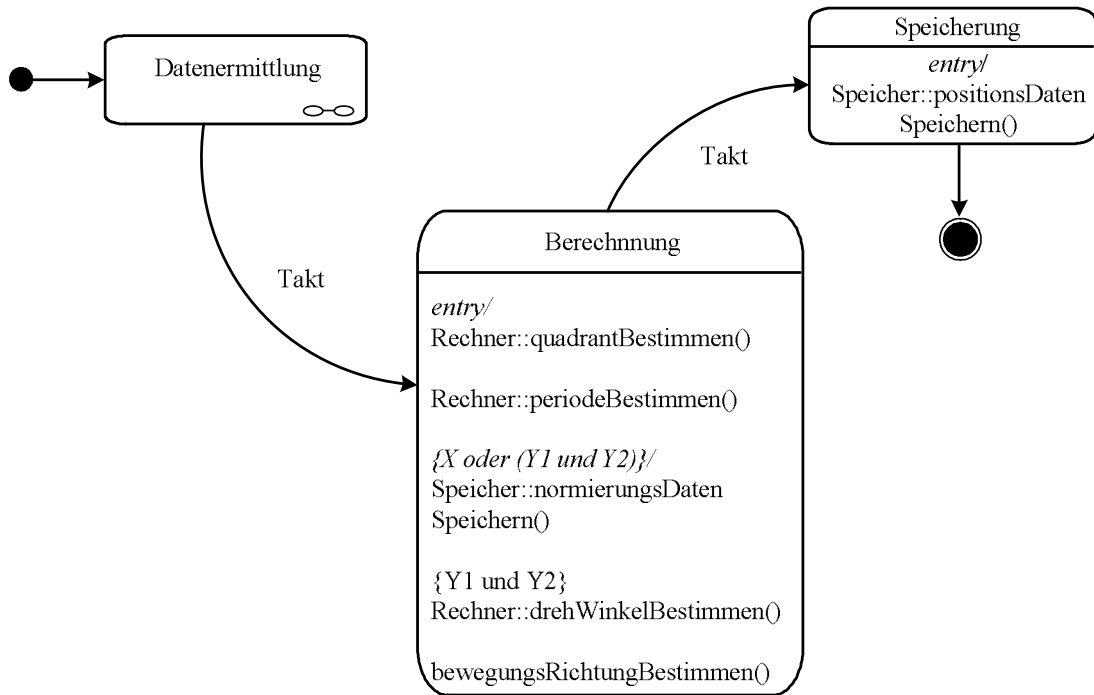


Abbildung 85: Modellierung des Messsystems mittels Zustandsdiagramm

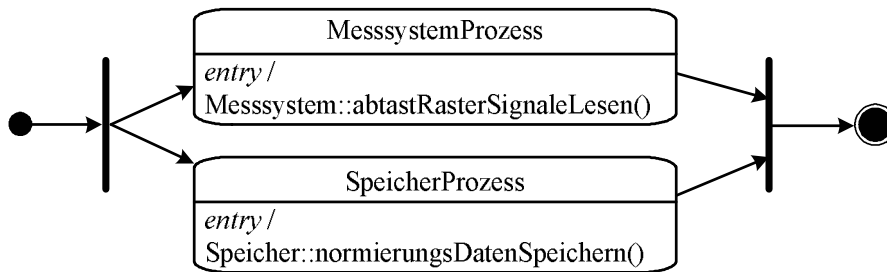


Abbildung 86: Verfeinerung des Zustandes „Achse Bewegung“

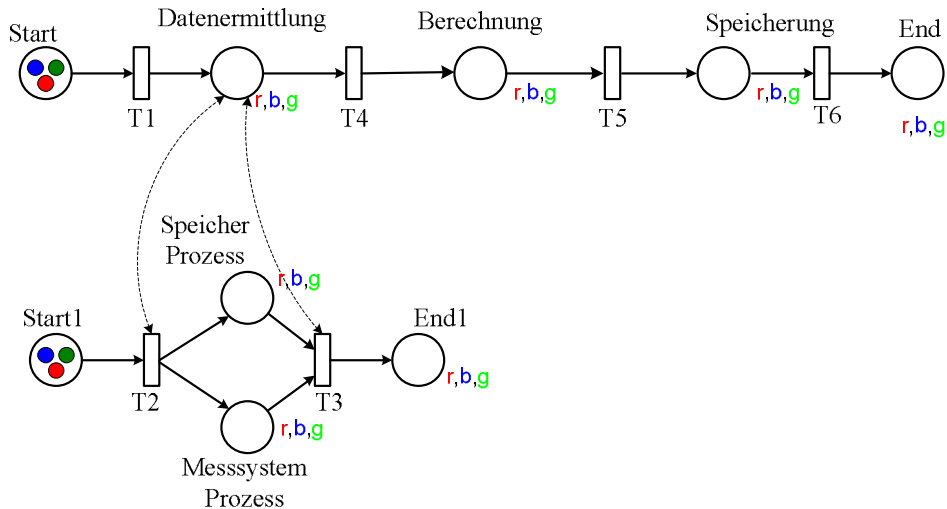


Abbildung 87: Nach der Transformation entstandenes Gefärbtes Petri-Netz

Tabelle 11: Zusätzliche Informationen zu dem Zustandsdiagramm

Farben:	X, Y1, Y2 sind entsprechend durch r,b,g in Petri-Netz dargestellt	
Platz	Kapazität	Aktion(en)
Start	X, Y1, Y2 <i>Anfangsmarkierung:</i> X, Y1, Y2	
Datenermittlung	X, Y1, Y2	
Speicher Prozess		Speicher::normierungsDatenSpeichern()
Messsystem Prozess		Messsystem::abtastRasterSignaleLesen()
Berechnung	X, Y1, Y2	<i>entry/</i> Rechner::quadrantBestimmen() Rechner::periodeBestimmen() <i>X oder (Y1 und Y2)/</i> Speicher::normierungsDatenSpeichern() Y1 und Y2 Rechner::drehWinkelBestimmen() bewegungsRichtungBestimmen()
Speicherung	X, Y1, Y2	Speicher::positionsDatenSpeichern()
End	X, Y1, Y2	
Transition	Transitionsfunktion	Ereignis
T1	-	-
T2	-	-
T3	-	-
T4	-	Takt
T5	-	Takt
T6	-	-

Es ist wichtig, bei den Transformationen, bei denen als Ausgangsdiagramm ein Zustandsdiagramm genutzt wird, folgendes zu berücksichtigen [Slav 2003]:

- die Struktur und Anzahl von Zuständen im Zustandsdiagramm hat keinen direkten Einfluss auf die Struktur des Aktivitätsdiagramms;
- alle im entstandenen Petri-Netz erwähnten Aktionen und Aktivitäten treten im generierten Aktivitätsdiagramm als Aktivitäten auf;
- wenn mehrere Aktionen in einem Zustand aufgerufen werden, dann werden sie als sequentiell durchgeführte Aktivitäten im Aktivitätsdiagramm erscheinen;
- alle Informationen über die „Färbung“ des Aktivitätsdiagramms ergeben sich aus farbabhängigen Bedingungen, Zustandskapazitäten und Transitionsfunktionen.

- Die Transformation von einem Zustandsdiagramm ins Sequenzdiagramm über das Petri-Netz ist nur dann möglich, wenn alle im Zustandsdiagramm vorhandenen Informationen über die Aktionen, Aktivitäten und Bedingungen im Petri-Netz erhalten

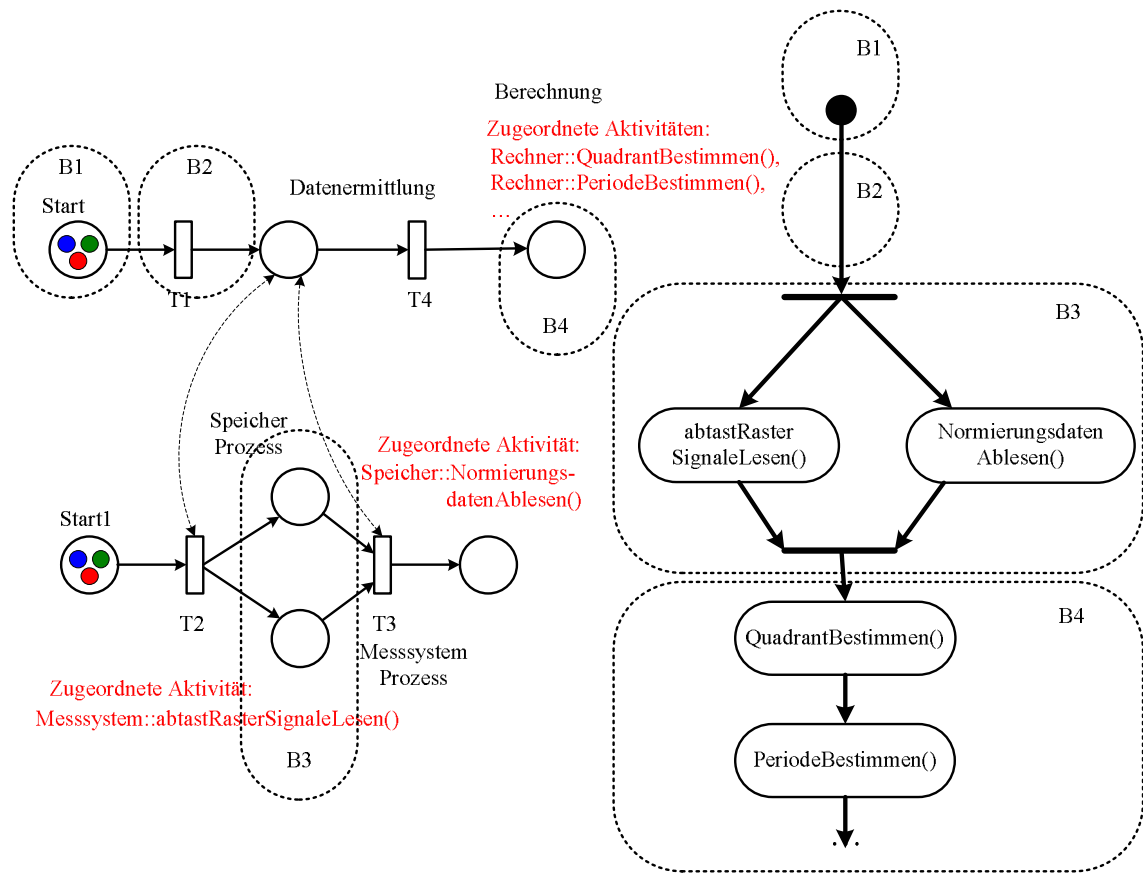


Abbildung 88: Transformation von Bausteinen des Petri-Netzes in das Aktivitätsdiagramm

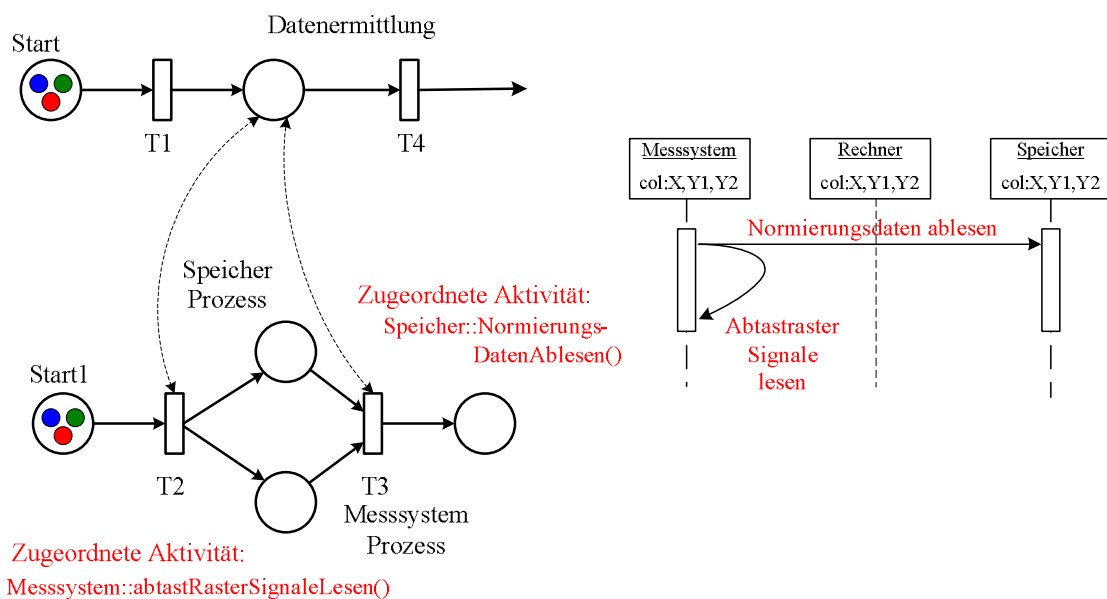


Abbildung 89: Transformation von Bausteinen des Petri-Netzes in das Sequenzdiagramm

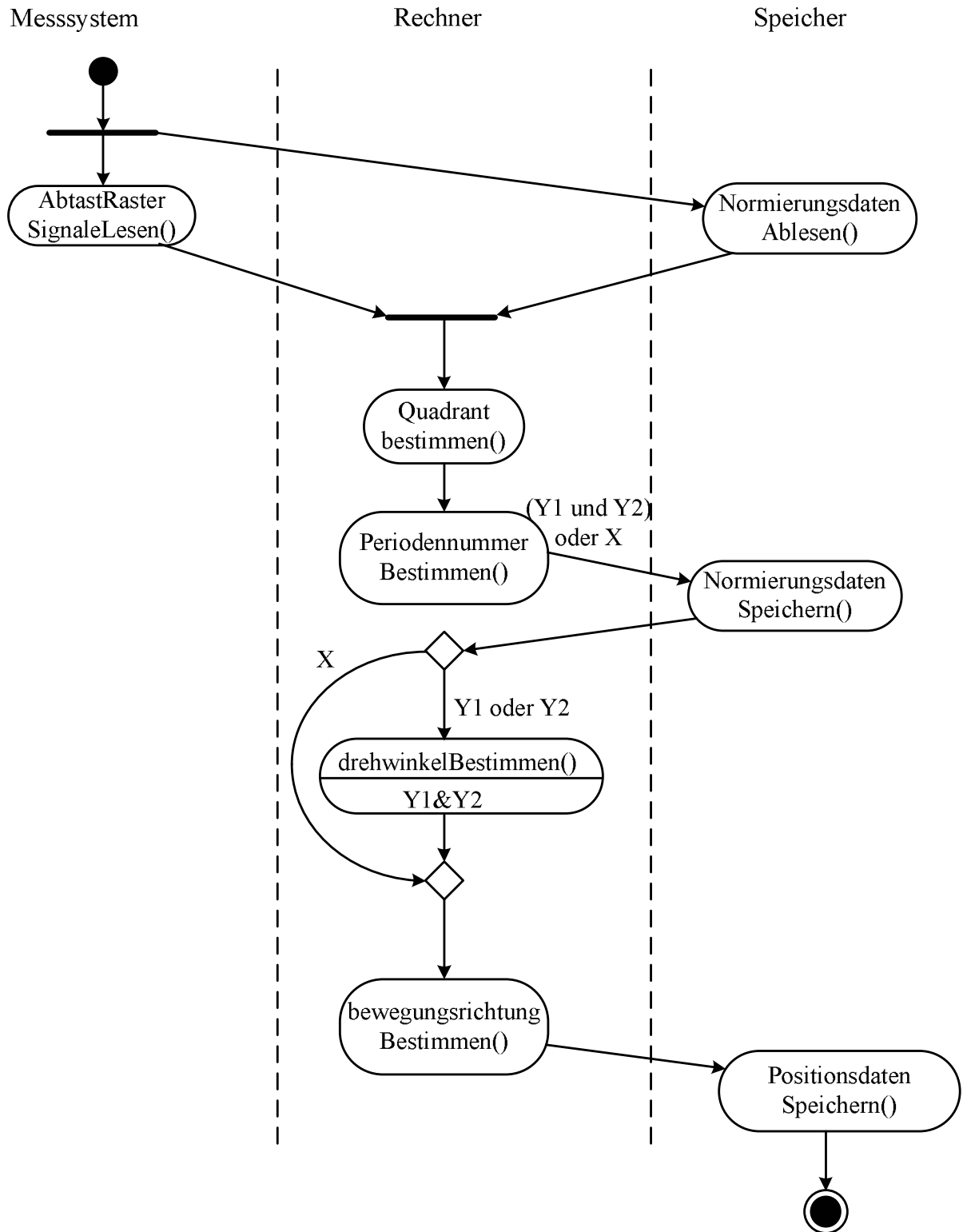


Abbildung 90: Gefärbtes Aktivitätsdiagramm des Messsystems

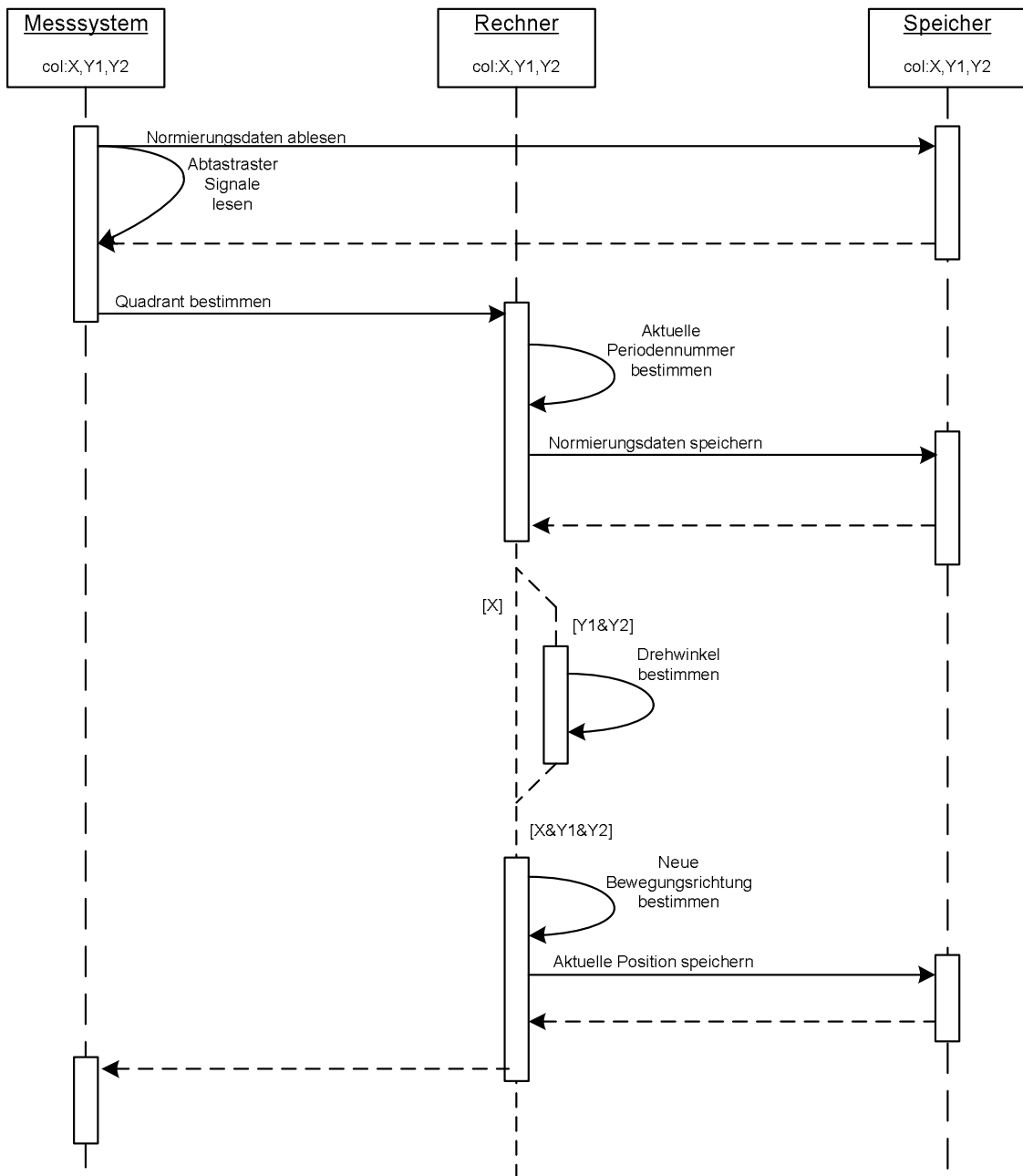


Abbildung 91: Gefärbtes Sequenzdiagramm des Messsystems

bleiben. Das heißt, diese Informationen müssen im Petri-Netz zusätzlich in einer textuellen Form gespeichert werden;

- die Anzahl von Objekten im entstandenen Diagramm ist gleich der Anzahl von Objekten in einem Zustandsdiagramm, welche Methoden aufrufen bzw. Aktionen durchführen;
- die Zuordnung von Objekten und Klassen ergibt sich aus der entsprechenden Zuordnung im Zustandsdiagramm;
- alle im entstandenen Petri-Netz erwähnten Aktionen und Aktivitäten treten im generierten Sequenzdiagramm als Methodenaufrufe bzw. Botschaften auf.

Die nächsten Abbildungen 92, 93, 94[Riab 2003] zeigen, wie jetzt das entstandene Aktivitätsdiagramm in ein Gefärbtes Petri-Netz überführt werden kann. Zuerst wird dieses Diagramm

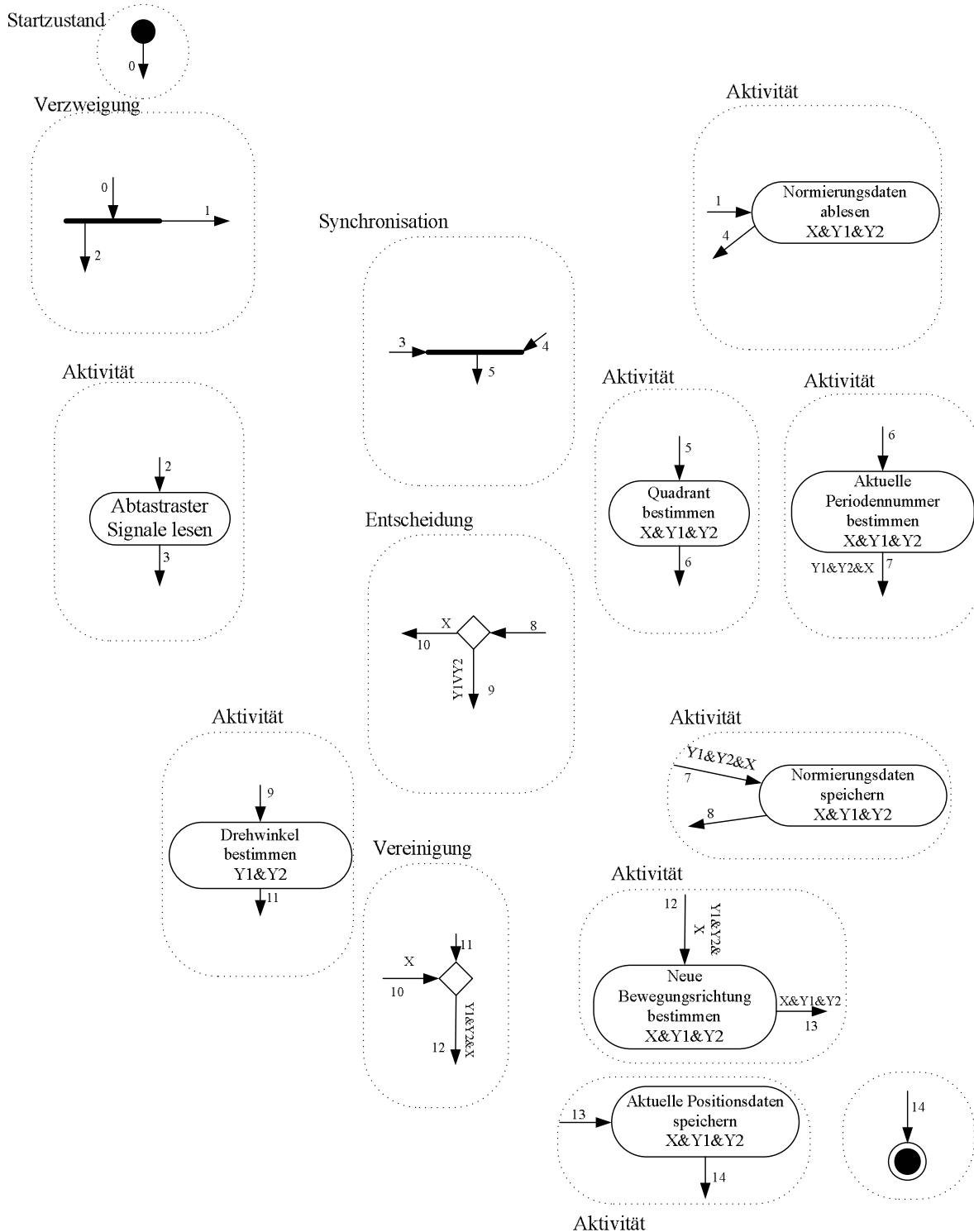


Abbildung 92: Aufspaltung eines Sequenzdiagramms in mehrere Bausteine

in Bausteine aufgespaltet. Danach werden diese Bausteine in Petri-Netz-Bausteine entsprechend der oben aufgezählten Regeln transformiert (Abbildung 93). Zum Schluss werden die Bausteine so zusammengesetzt, dass ein Gefärbtes Petri-Netz entsteht (Abbildung 94).

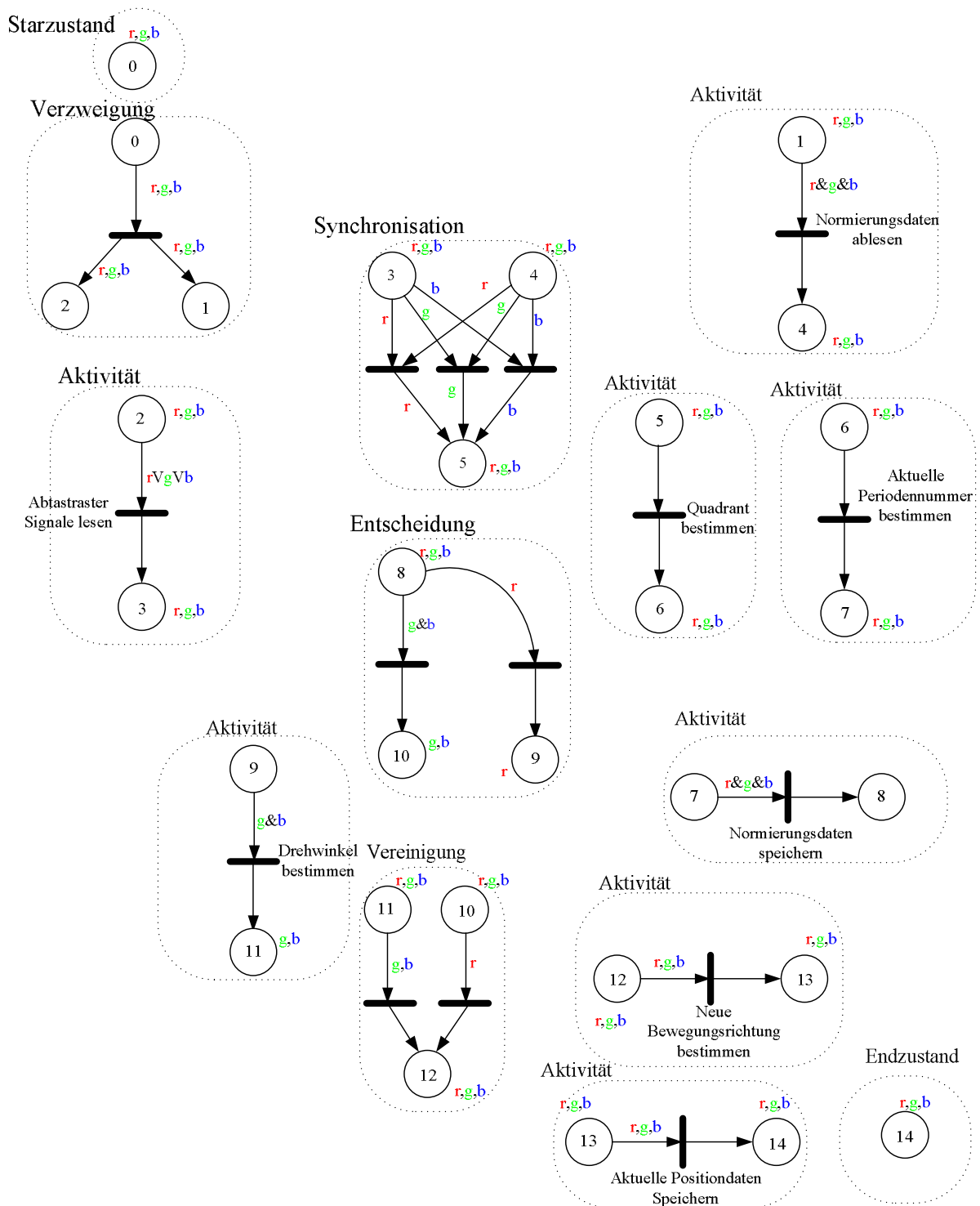


Abbildung 93: Ersetzung der Aktivitätsdiagrammbausteine durch Bausteine des Petri-Netzes

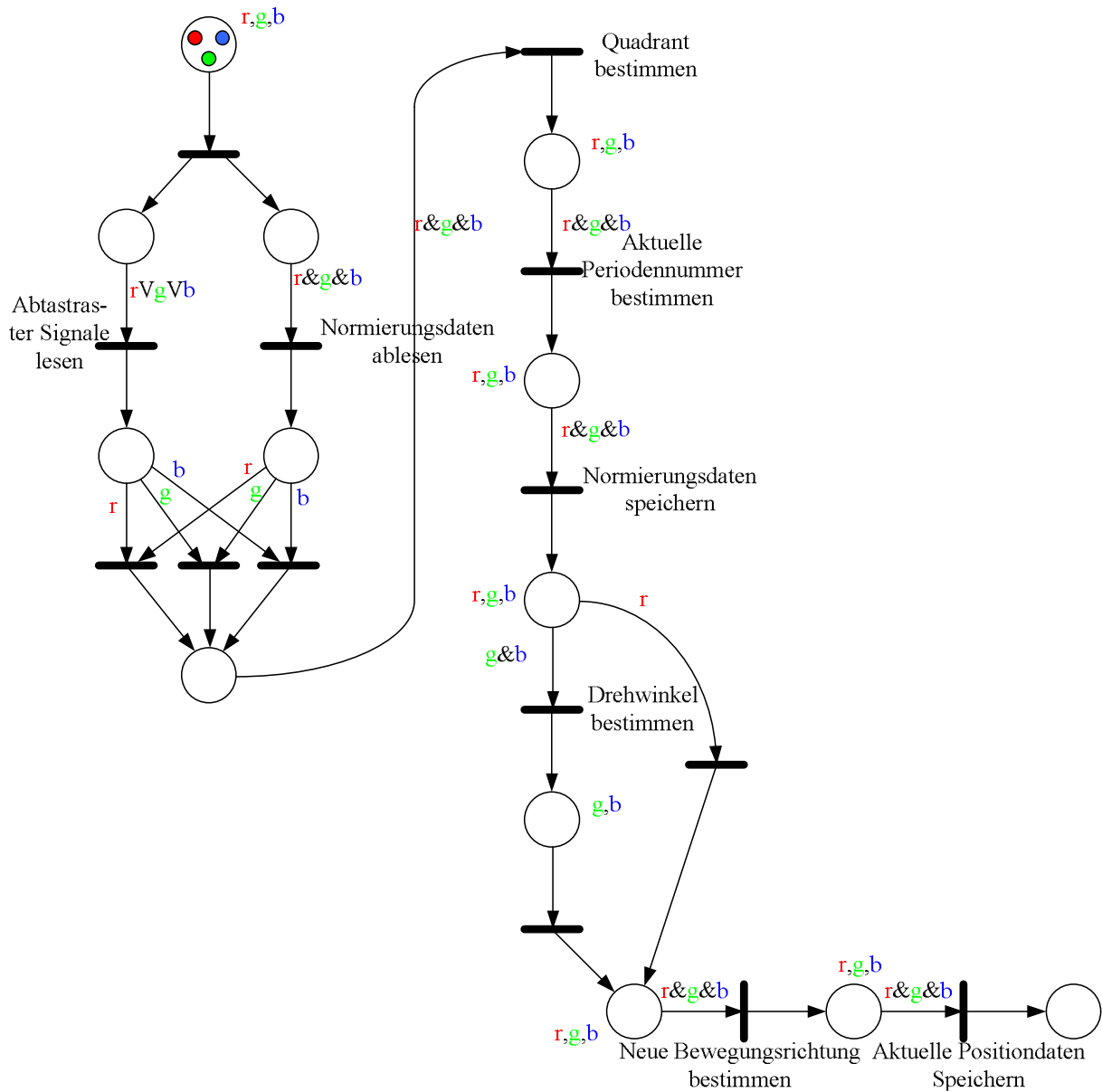


Abbildung 94: Aus dem Aktivitätsdiagramm entstandenes Gefärbtes Petri-Netz

Dieses Petri-Netz als Zwischennotation wird zum Zweck der Verifikation des entwickelten Modells eingesetzt.

Die Petri-Netz-Bausteine werden, wie in den weiteren Abbildungen gezeigt, in Bausteine der verschiedenen Diagramme nach den oben aufgezählten Regeln transformiert. Zum Schluss erfolgt die Zusammensetzung der entstandenen Bausteine, so die verschiedenen gefärbten Diagramme entstehen (Abbildungen 95, 96, 97 [Riab 2003]).

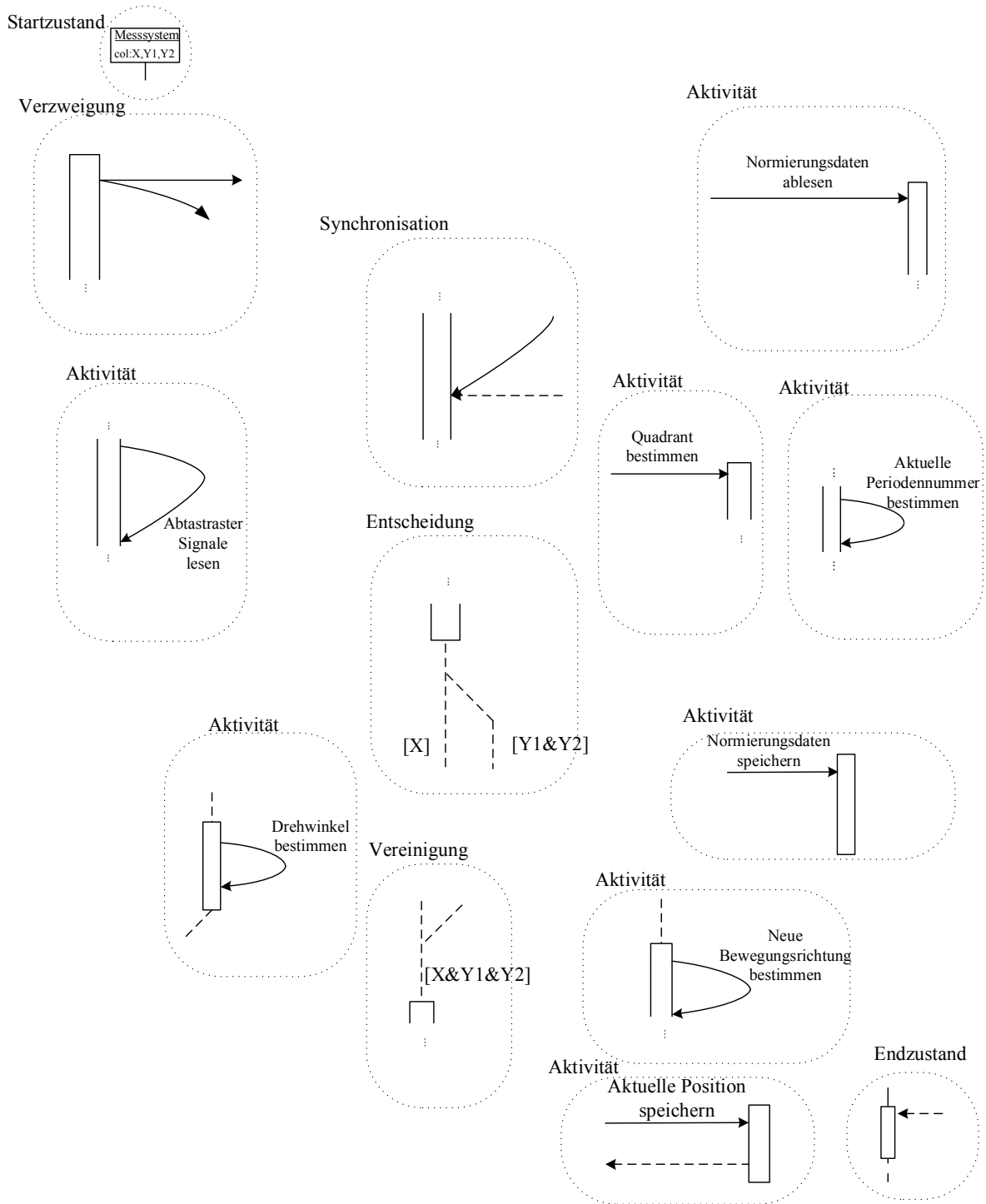


Abbildung 95: Transformation der Gefärbten Petri-Netz-Bausteine in Bausteine des gefärbten Sequenzdiagramms

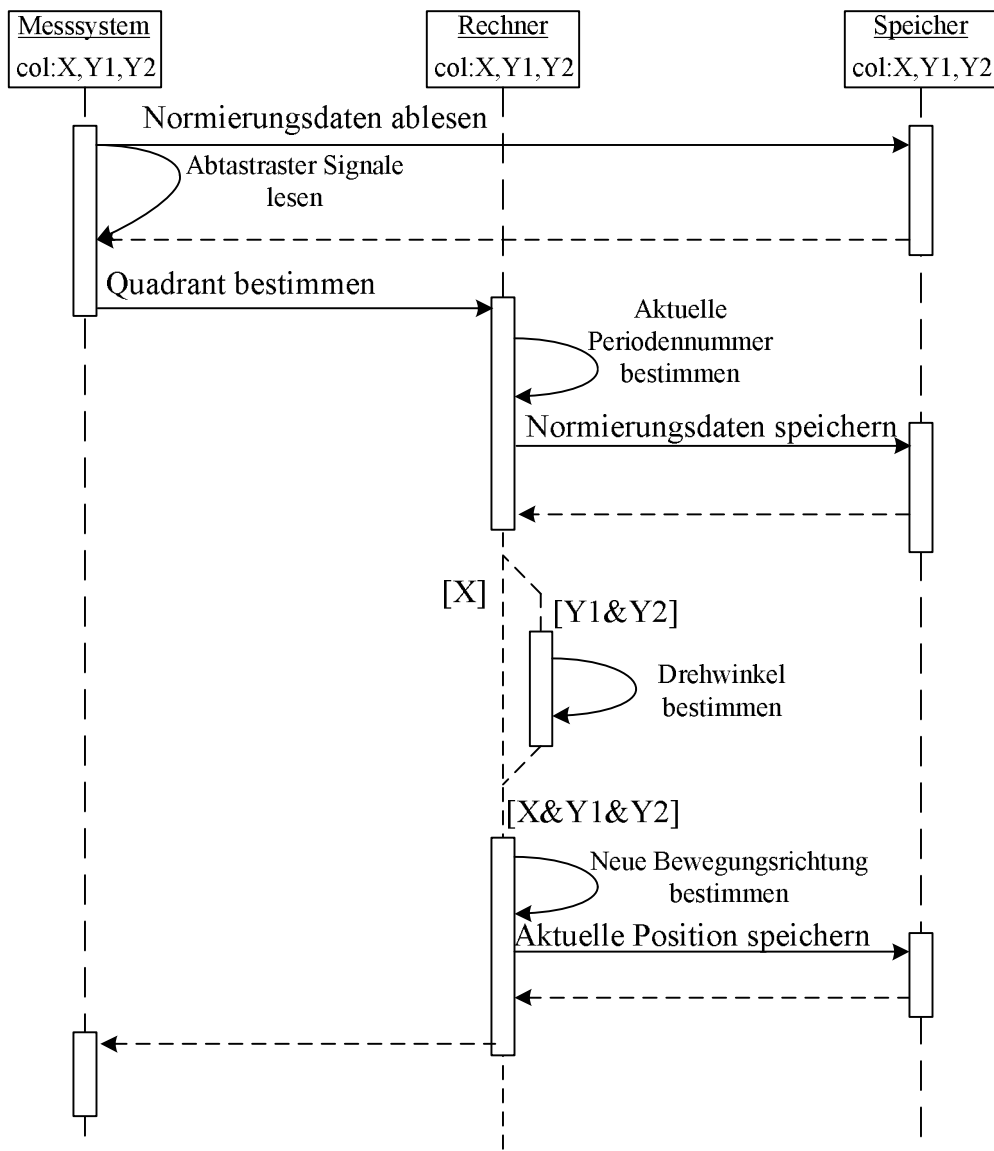


Abbildung 96: Aus dem Aktivitätsdiagramm entstandenes gefärbtes Sequenzdiagramm

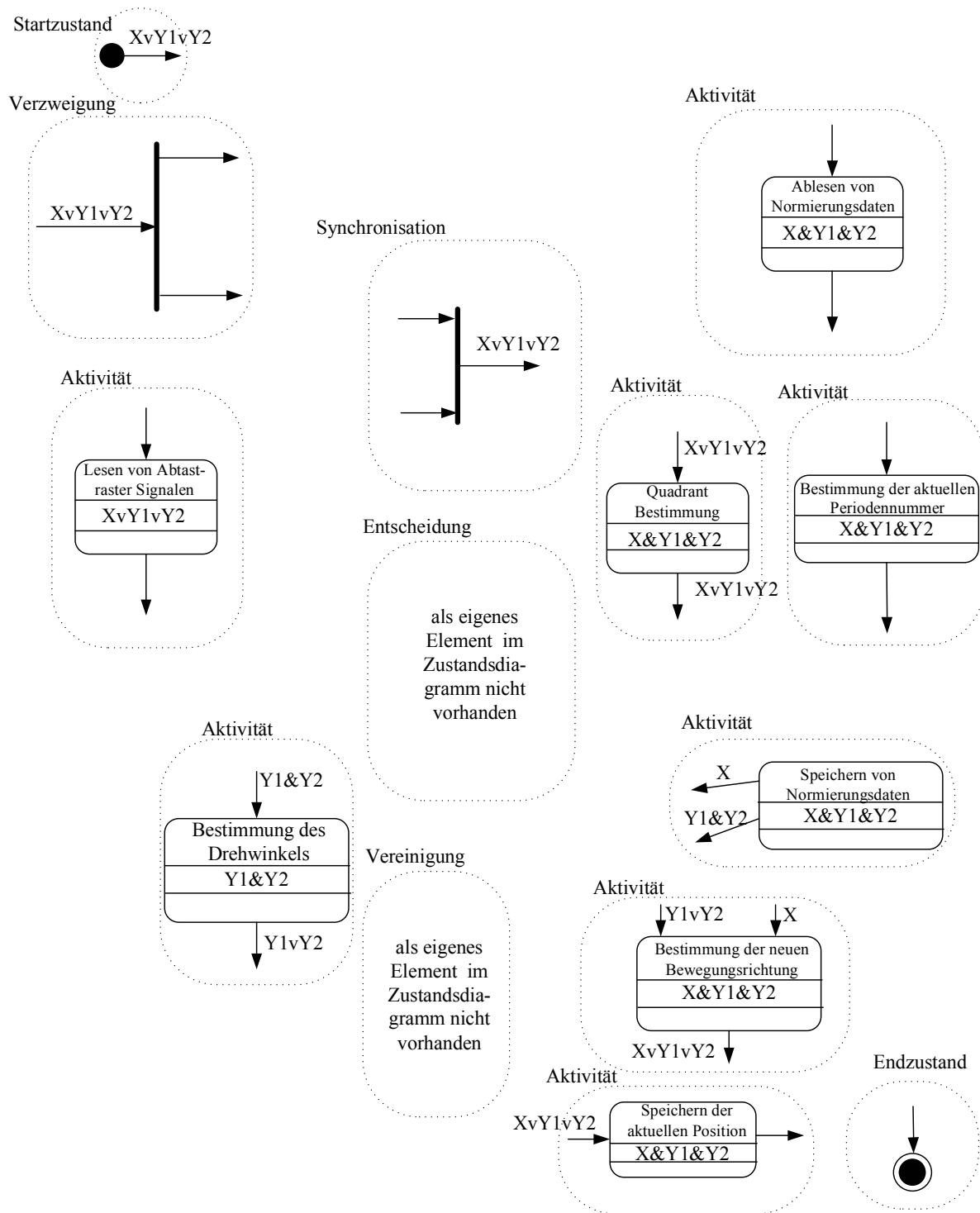


Abbildung 97: Transformation der aus dem Aktivitätsdiagramm entstandenen Petri-Netz-Bausteine in das gefärbte Zustandsdiagramm

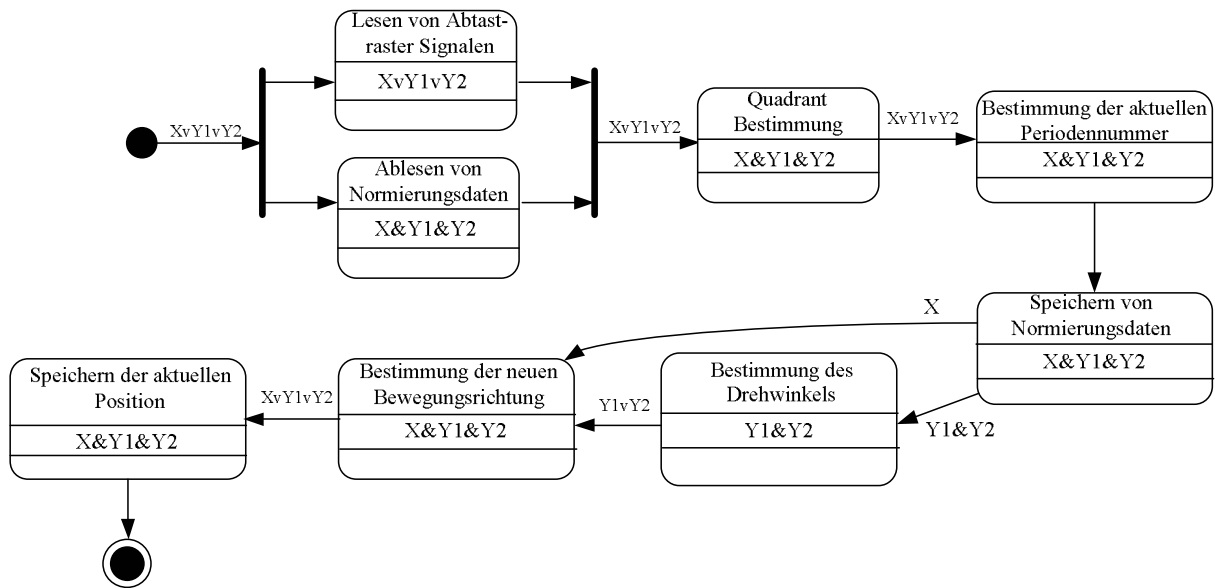


Abbildung 98: Aus dem Aktivitätsdiagramm entstandenes gefärbtes Zustandsdiagramm.

5.6 Toolrealisierung

Die Konvertierung von gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen in Gefärbte Petri-Netze wird durch drei Tools, die von gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen in Gefärbte Petri-Netze konvertieren, unterstützt. Für diese Tools wurde für die Eingabe der drei gefärbten Diagramme Rational Rose [Rational] verwendet. Rational Rose ist ein CASE-Werkzeug zur visuellen Modellierung und ist gut bekannt und verbreitet, da die Entwickler dieses Produktes, Grady Booch, Ivar Jacobsen und James Rumbaugh, die Urheber der Unified Modeling Language, sind. Die diesem Werkzeug zugrunde liegenden Notationen entsprechen deshalb denen der UML. Dabei wird mit Hilfe von Rational Rose zuerst ein normales Zustands-, Aktivitäts- oder Sequenzdiagramm erstellt. Erst danach werden diese Diagramme um die neue Eigenschaft – die Farbe – erweitert, welche als zusätzlicher Parameter bestimmter Elemente im Dokumentationsfeld erscheint. Z.B. können im Sequenzdiagramm die Instanzen und Nachrichten Träger von Farben sein, deshalb bekommen sie im dokumentierenden Teil zusätzliche Felder, in denen die Farbinformation beschrieben wird. Es wird das Feld „Eigenschaften/Dokumentation“ gewählt und dort wird „colour“ mit der Liste der definierten, zu dieser Nachricht gehörenden Farben eingetragen. Dabei kann für die Nachrichten auch colour sowohl für die Senderseite als auch für die Empfängerseite angegeben werden, falls diese unterschiedlich sind. Zum Schluss werden alle diese Angaben einheitlich in eine Petal-Datei mit Erweiterung *ptl* exportiert. Diese Dateien beschreiben vollständig gefärbte Zustands-, Aktivitäts- und Sequenzdiagramme und werden dann als Eingabeformat für die Konvertierungstools benutzt (Abbildungen 99, 100 [Riab 2001]).

Da nach umfangreichen Recherchen kein passendes Petri-Netz-Tool mit der Möglichkeit, die Gefärbten Petri-Netze aus einem Import-Dateiformat mit allen benötigten Angaben einzulesen, gefunden wurden, wurde als Ausgabeformat für die Beschreibung der nach der Konver-

tierung entstandenen Gefärbten Petri-Netze ein auf [LyMa] basierendes Textformat benutzt. Das führt zu einer hohen Flexibilität bei der Arbeit mit diesen XML-basierten Daten, da diese sich gut für den Datenaustausch mit verschiedenen Anwendungen eignen. Dabei können sie auch mittels des Tools XLTS in ein anderes Format transformiert werden.

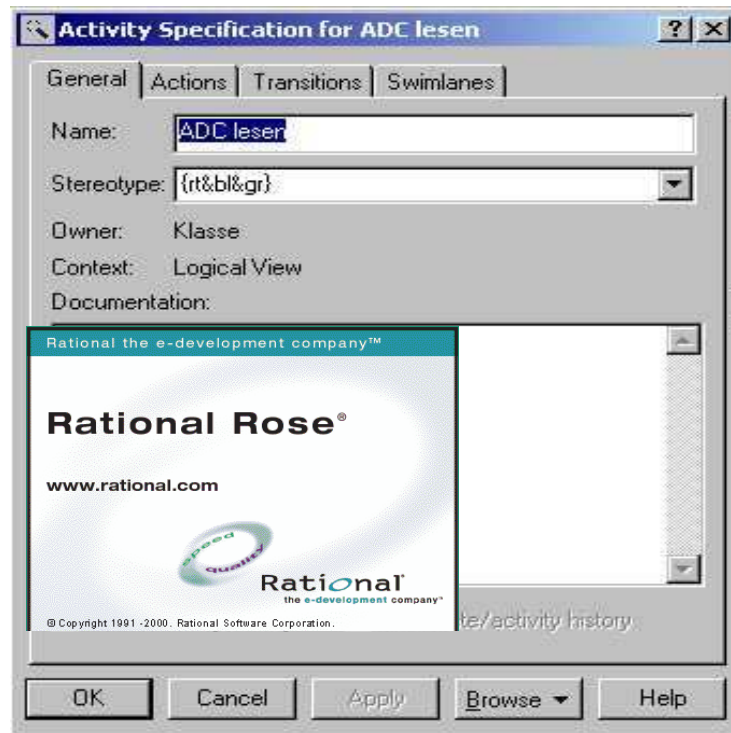


Abbildung 99: Erstellen der Petal-Dateien

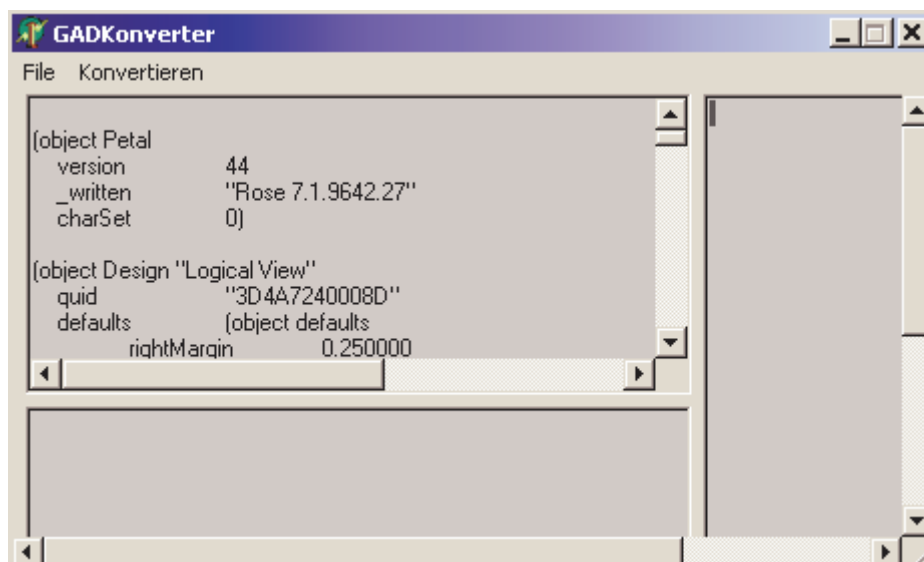


Abbildung 100: Einlesen einer Petal-Datei in einen Konverter

Die aus der Petal-Datei ausgelesenen Daten werden entsprechend der schon in den früheren Unterkapiteln beschriebenen Transformationsregeln in Gefärbte Petri-Netze überführt.

Im Ausgabeformat werden folgende Tags zu Beschreibung von Gefärbten Petri-Netzen benutzt:

<CPN>	bezeichnet eine CPN-Beschreibung
<PLACE>	Bezeichner eines Platzes
<TRANSITION>	Bezeichner einer Transition
<ARC>	Bezeichner einer Kante
<TEXT>	Bezeichner einer zusätzlichen textuellen Anmerkung
<CAPACITY>	beschreibt die Kapazitäten von Plätzen
<INITMARK>	beschreibt die Anfangsmarkierung von Plätzen
<ARCWEIGHT>	Vielfachheit einer Kante bzw. Kantengewichte einer gefärbten Kante
<COLCONV>	Farbkonvertierungsfunktion einer Transition, sie beschreibt die Zuordnung von Kantengewichten der Vor- und Nachkanten

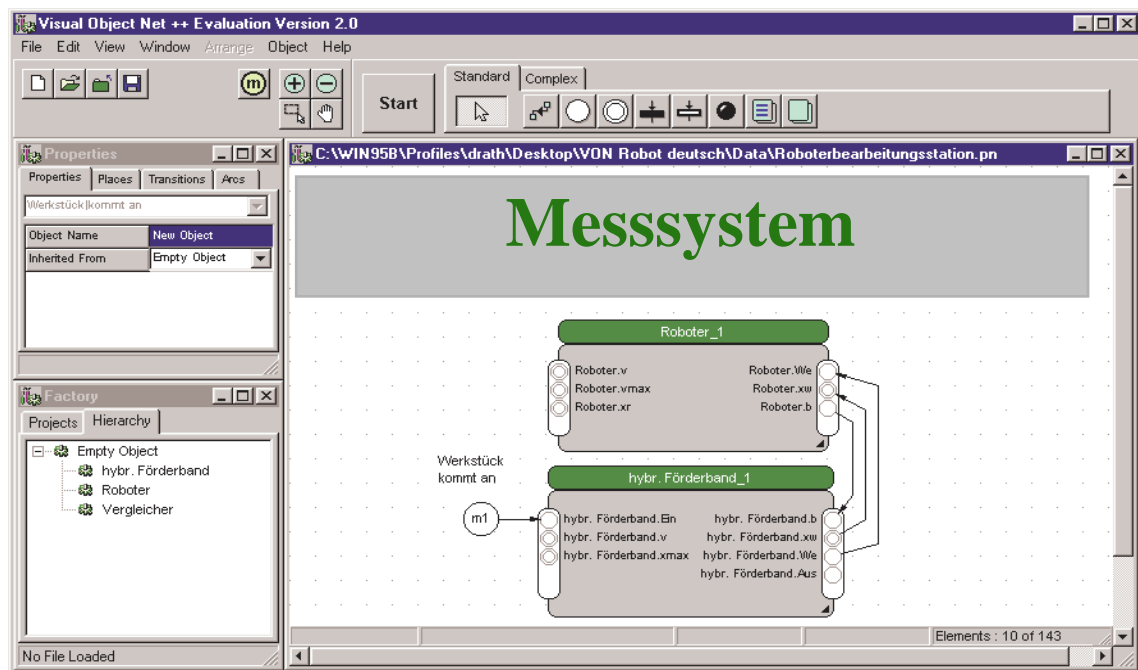


Abbildung 101: Überprüfung eines Modells (Messsystem) in Visual Object Net++ [Drath 99]

Die durch diese Transformationsverfahren gewonnenen Gefärbten Petri-Netze können jetzt weiter zur Überprüfung des ursprünglichen Modells auf dem Niveau des Petri-Netzes eingesetzt werden. Z.B. zeigt die Abbildung 101 das Werkzeug Visual Object Net++, mit Hilfe dessen dann Simulationen durchgeführt werden. Zur zeitlichen und zur strukturellen Analyse wird angewendet.

Kapitel 6

Verifikation von gefärbten Zustands-, Aktivitäts- und Sequenzdiagrammen

Um die Korrektheit des Entwurfes von eingebetteten Echtzeitsystemen und die Einhaltung von Anforderungen nachweisen zu können, wird während des Entwurfsprozesses die Verifikation des zu entwickelnden Systems notwendig. Da in dieser Arbeit als Zwischennotationsmittel Petri-Netze eingesetzt werden, für die eine große Palette an formalen Verifikationsmethoden entwickelt wurde, sollen hier die Verifikationsmethoden von Petri-Netzen weiter betrachtet werden. In diesem Kapitel wird gezeigt, wie sich die Verifikationsmethode für Zeitintervall-Petri-Netze auf Gefärbte Zeitintervall-Petri-Netze erweitern lässt. Dabei geht es um die Gefärbten Petri-Netze, die bei der Transformation aus gefärbten Sequenz-, Zustands- und Aktivitätsdiagrammen entstanden sind, welche zur Modellierung von eingebetteten Echtzeitsystemen verwendet wurden.

Am Anfang wird in diesem Kapitel eine Definition von Gefärbten Zeitintervall-Petri-Netzen gegeben und danach wird eine Verifikationsmethode für die dynamischen Aspekte vorgestellt, welche sowohl Aussagen über die Erfüllung von zeitlichen Restriktionen als auch über die Erreichbarkeit von Markierungen, Lebendigkeit, Konfliktfreiheit, Beschränktheit und den Nachweis von möglichen dynamischen Konflikten liefert. Zum Schluss wird ein Tool, welches diese Methode unterstützt, kurz dargestellt.

6.1 Gefärbte Zeitintervall-Petri-Netze

Ein gefärbtes Zeitintervall-Petri-Netz wird in dieser Arbeit wie folgt definiert:

Ein Tupel $CTN = (P, T, F, V, K, C, m_o, \zeta, I, f_t)$ ist ein Gefärbtes Zeitintervall-Petri-Netz, wenn:

1. $CN = (P, T, F, V, K, C, m_o, \zeta)$ ein Gefärbtes Petri-Netz ist
2. I : eine Menge von Zeitintervallen mit Elementen (t_{\min}, t_{\max})
3. $f_t = S \rightarrow I$ eine schaltmodusabhängige Zeitintervall-Bewertung der Transitionen.

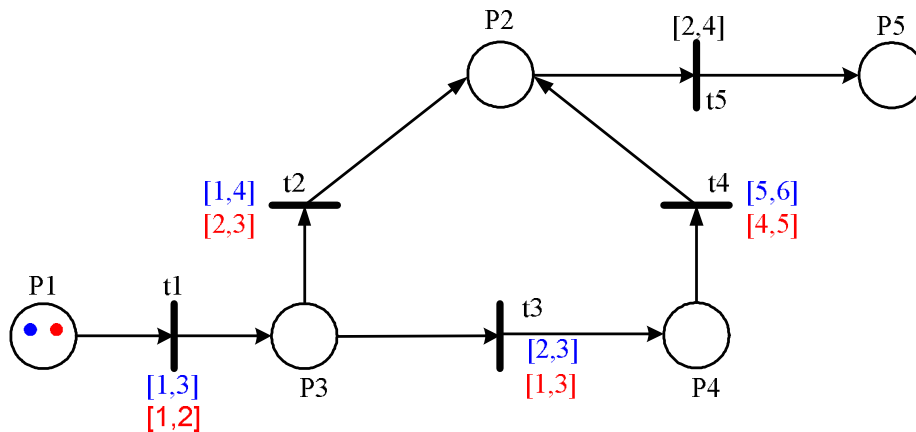


Abbildung 102: Beispiel für ein Gefärbtes Zeitintervall-Petri-Netz

In Abbildung 102 [Glas 2002] wird ein Beispiel für ein Gefärbtes Zeitintervall-Petri-Netz gezeigt. Da die mathematische Darstellung nicht sehr anschaulich ist, wird hier wie bei Gefärbten Petri-Netzen eine tabellarische Form der Notation für ζ und I benutzt.

Im vorliegenden Fall für die Transition t1 die Konvertierungsfunktionen und zu der Transition gehörige Zeitintervalle für verschiedene Farben (Abbildung 103).

	k1	k2	
t1:	r	r	[1,2]
	b	b	[1,3]

Abbildung 103: Tabellarische Darstellung der Konvertierungsfunktion mit Zuordnung von Zeitintervallen

Ein Zustand in einem Gefärbten Zeitintervall-Petri-Netz wird aus einer Markierung und einem Vektor von schon vergangenen Zeiten beschrieben:

$$Z = ((m_1, m_2, m_3, \dots), \left(\begin{array}{l} \{ \text{Schaltmgl.1 Zeit1, Schaltmgl.2 Zeit2, } \} \\ \{ \text{Schaltmgl.1 Zeit1, Schaltmgl.2 Zeit2, } \} \end{array} \right))$$

Es wird dabei zu jeder Transition für jede Schaltmöglichkeit (Schaltmgl.) die Zeitangabe zugeordnet.

Im Beispiel Abbildung 102 entsteht folgender Startzustand:

$$Z = ((\{1r,1b\}, \{0r,0b\}, \{0r,0b\}, \{0r,0b\}, \{0r,0b\}, \{0r,0b\}), \begin{pmatrix} \{\{1r\}0, \{1b\}0\} \\ \{\{1r\}\#, \{1b\}\#\} \\ \{\{1r\}\#, \{1b\}\#\} \\ \{\{1r\}\#, \{1b\}\#\} \\ \{\{1r\}\#, \{1b\}\#\} \end{pmatrix})$$

Den Wert # bekommen alle nicht schaltfähige Transitionen.

Ein Zustand in einem Gefärbten Zeitintervall-Petri-Netz wird folgend definiert [Glas 2002]:

Wenn $CTN = (P, T, F, V, K, C, m_o, \zeta, I)$ ein Gefärbtes Zeitintervall-Petri-Netz ist, dabei $t: T \rightarrow Q_0^+ \cup \{\#\}$ und m_m eine Markierung im CTN ist, die als Matrix dargestellt wird, dann heißt ein Paar $z_C := (m_m, tm)$ ein Zustand im CTN, wenn

1. $\forall t (t \in T \wedge t^- \leq m_m) \rightarrow tm(t) \leq lft_m(t)$
2. $\forall t (t \in T \wedge t^- > m_m) \rightarrow tm(t) = \#$.

Um die gefärbten Zeitintervall-Petri-Netze verifizieren zu können, werden hier angelehnt an normale Zeitintervall-Petri-Netze Zustandsklassen definiert.

Dabei ist ein Zustand $z_{C0} := (m_{m0}, tm_0)$, wobei

$$tm_0 := \begin{cases} 0, & \text{wenn } t^- \leq m_{m0} \\ \#, & \text{sonst} \end{cases}$$

gilt, der Initialzustand oder Startzustand im gefärbten Zeitintervallen-Petri-Netz.

Die Situation im Gefärbten Zeitintervall-Petri-Netz verändert sich, wenn eine Transition schaltet oder die Zeit sich verändert. Der Zustand $z_C = (m_m, tm)$ geht beim Schalten der Transition \hat{t} in den Zustand $z_C' = (m_m', tm')$ über, wobei $m_m' = m_m + \Delta \hat{t}$ und

$$tm'(t) = \begin{cases} \#, & \text{wenn } t^- > m_m \\ tm(t), & \text{wenn } t^- \leq m_m \wedge t^- \leq m_m \wedge Ft \cap F\hat{t} = \emptyset \\ 0, & \text{sonst} \end{cases}$$

Der Zustand $z_C = (m_m, tm)$ verändert sich durch den Zeitablauf $\tau \in Q_0^+$ in den Zustand

$z_C' = (m_m', tm')$, d.h. $z_C \xrightarrow{\tau} z_C'$, wenn folgendes gilt:

1. $m_m' = m_m$
2. $\forall t (t \in T \wedge tm(t) \neq \#) \rightarrow tm(t) + \tau \leq lft_m(t)$ und
3. $tm'(t) := \begin{cases} tm(t) + \tau, & \text{wenn } t^- \leq m_m \\ \#, & \text{sonst} \end{cases}$

Für die Verifikationsmethode benötigen wir Transitionssequenzen des analysierten Gefärbten Intervall-Petri-Netzes, welche als kanonische Parameterdarstellung $\delta(w)$ definiert werden können.

Dabei wird für ein $CTN = (P, T, F, C, K, V, m_0, \zeta, I)$ eine partielle Abbildung folgendermaßen definiert:

$$\delta : T^* \rightarrow R(m_{m_0}) \times SUM^{T,C} \times P(BED_C).$$

Als Erweiterung von der Definition aus [Popo 89] kommt hier noch eine Farbabhängigkeit dazu.

Für den ersten Schritt $\delta_C(e) := [m_{m_0}, \Sigma_{C_0}, B_{C_0}]$ gilt folgendes:

1. $m_{m_e} = m_{m_0}$

2.
$$\Sigma_{C_e}(t) := \begin{cases} x_0, & \text{wenn } t^- \leq m_{m_e} \\ \#, & \text{sonst} \end{cases}$$

3. $B_{C_e} := \{A^2 \Sigma_{C_e}(t) v_t \mid t^- \leq m_{m_e}\}$

Für jeden weiteren Schritt $\delta_C(\alpha\hat{t}) := [m_m', \Sigma_C', B_C']$ gilt:

1. $m_m' = m_m + \Delta(\hat{t})$

2. j ist ein kleinster Index und die Variable x_j tritt weder in Σ noch in B auf und j ist auch größer als alle Indexes, deren Variablen in Σ oder in B auftreten.

3.
$$\Sigma_C'(t) := \begin{cases} \#, & \text{wenn } t^- \leq m_m \\ x_j, & \text{wenn } (t^- \not\leq m_m \wedge t^- \leq m_m') \vee (t^- \leq m_m \wedge t^- \leq m_m') \\ \Sigma_C(t) + x_j, & \text{sonst} \end{cases}$$

4. $B_C' := B_C \cup \{A^2 u_t(\Sigma_C(\hat{t}))\} \cup \{A^2 \Sigma_C'(t) v_t \mid t^- \leq m_m'\}$

Für die Zustandsklassen C_0 und C ist die Abbildung $\delta(w) = [m, \Sigma, B]$ auch die kanonische Parameterdarstellung von C . Dabei können auch die Zustandsklassen C durch die Abbildung $\delta(e)$ dargestellt werden.

6.2 Die Verifikationsmethode

Für die Durchführung der Verifikation des Gefärbte Intervall-Petri-Netzes ist eine Schaltsequenz für die Transitionen mit Angabe von Zeitintervallen nötig. Durch die Analyse mit Hilfe der auf Farben erweiterten Verifikationsmethode von Popova [Popo 97] werden die zeitlichen Grenzwerte dieser Schaltsequenz optimiert.

Für die Anwendung dieser Methode wird erst eine Schaltsequenz für das zu untersuchende Gefärbte Zeitintervall-Petri-Netz festgelegt, welche später überprüft werden soll. Als weiteres werden Grenzwerte für Zeitvariable gesucht und das zum Schluss entstandene Ungleichungs-

system wird mit Hilfe von Methoden der linearen Optimierung gelöst. Nach der Festlegung einer Schaltsequenz beginnt man mit dem Startzustand. Alle in diesem Zustand schaltfähigen Transitionen bekommen als zeitliche Variable x_1 zugeordnet, alle anderen #. Durch das Schalten von der als erstes angegebenen Sequenz schaltfähiger Transitionen bekommt man die nächste Zustandsklasse. Dabei erhalten alle neu aktivierten Transitionen jetzt die Variable x_2 als Zeitbewertung und alle noch schaltfähigen Transitionen x_1+x_2 . Durch Schalten der nächsten Transition geht man wieder zu einer neuen Zustandsklasse über. Diese Schritte werden so lange durchgeführt, bis die alle Transitionen der Sequenz geschaltet haben oder die Nichtausführbarkeit dieser Sequenz festgestellt ist.

In der Abbildung 104 [Roth 2002] ist ein Gefärbtes Petri-Netz dargestellt. Beispielhaft wird hier die Transitionsequenz $\sigma=(t1(r,g \rightarrow r)t2t3t1)$ untersucht.

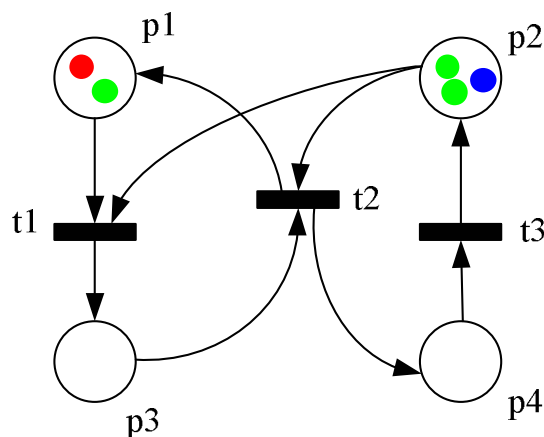


Abbildung 104: Gefärbtes Petri-Netz

Die Konvertierungsfunktion und die Zuordnung von Zeitintervallen für dieses Beispiel werden in tabellarische Form dargestellt:

	(p1,t1)	(p2,t1)	(t1,p3)	
t1:	r	g	r	[1,4]
	g	b	g	[0,8]
	(p3,t2)	(p2,t2)	(t2,p1)	(t2,p4)
t2:	r	g	r	b [2,5]
	g	b	g	g [1,3]
	(p4,t3)	(t3,p2)		
t3:	g	b [0,1]		
	b	2g,b [0,1]		

Mit der Einfügung von Variablen x_1, x_2, x_3, x_4 , welche die Schaltzeiten beschreiben, entsteht eine kanonische Parameterdarstellung $\delta(w)$. Die gesamte Transitionsequenz benötigt eine Zeit $X=x_1+x_2+x_3+x_4$ (Abbildung 105).

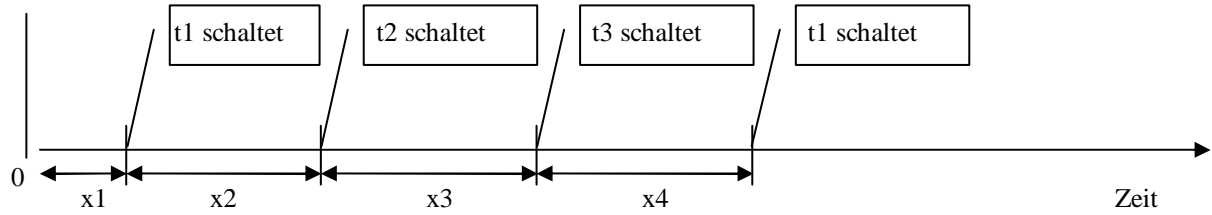


Abbildung 105: Zeiten für die Transitionsequenz

Es wird bei der Analyse mit dem Startzustand begonnen. In diesem Zustand bekommen alle schaltfähigen Transitionen die Zeitbewertung x_1 und alle andere den Wert #. Aus dem Zustand:

$$(\{r,g\}, \{2g,b\}, 0,0)$$

kann nur die Transition t_1 mit einer aus zwei ihrer Schaltmöglichkeiten schalten. Dabei werden aus beiden angegebenen Intervallen Beschränkungen für die Variable x_1 angelegt. Die Anfangsklasse wird jetzt folgendermaßen beschrieben:

$$C_0 = ((\{r,g\}, \{2g,b\}, 0,0), \begin{pmatrix} \{\{r,g\}x_1, \{g,b\}x_1\} \\ \{\{r,g\}\#, \{g,b\}\#\} \\ \{\{b\}\#, \{g\}\#\} \end{pmatrix}, 1 \leq x_1 \leq 3)$$

Der nächsten Zustand C_1 , welcher nach dem Schalten von t_1 entsteht, wird jetzt wie folgt beschrieben:

$$C_1 = ((\{g\}, \{g,b\}, \{r\}, 0), \begin{pmatrix} \{\{r,g\}\#, \{g,b\}x_1+x_2\} \\ \{\{r,g\}x_2, \{g,b\}\#\} \\ \{\{b\}\#, \{g\}\#\} \end{pmatrix}, \begin{matrix} 2 \leq x_1 \leq 3 \\ 2 \leq x_2 \leq 5 \\ 0 \leq x_1+x_2 \leq 8, \end{matrix})$$

wobei die Zeit x_2 eingefügt wird und die Transitionen t_1 und t_2 mit jeweils einer Schaltmöglichkeit schaltfähig sind.

Nach dem sequentiellen Schalten von den Transitionen t_2 und t_3 entstehen folgende Klassen:

$$C_2 = ((\{g,r\}, \{b\}, 0, \{b\}), \begin{pmatrix} \{\{r,g\}\#, \{g,b\}x_1+x_2+x_3\} \\ \{\{r,g\}\#, \{g,b\}\#\} \\ \{\{b\}x_3, \{g\}\#\} \end{pmatrix}, \begin{matrix} 2 \leq x_1 \leq 3 \\ 2 \leq x_2 \leq 5 \\ 0 \leq x_1+x_2 \leq 8 \\ 0 \leq x_1+x_2+x_3 \leq 8 \\ 0 \leq x_3 \leq 1 \end{matrix})$$

$$C_3 = ((\{g,r\}, \{2b,2g\}, 0,0), \begin{pmatrix} \{\{r,g\}x_4, \{g,b\}x_1+x_2+x_3+x_4\} \\ \{\{r,g\}\#, \{g,b\}\#\} \\ \{\{b\}\#, \{g\}\#\} \end{pmatrix}, \begin{matrix} 2 \leq x_1 \leq 3 \\ 2 \leq x_2 \leq 5 \\ 0 \leq x_1+x_2 \leq 8 \\ 0 \leq x_1+x_2+x_3 \leq 8 \\ 0 \leq x_3 \leq 1 \\ 0 \leq x_1+x_2+x_3+x_4 \leq 8, \end{matrix})$$

wobei die Transition t_1 wieder mit zwei verschiedenen Schaltmöglichkeiten schalten kann.

Ergebnisse:

Nach der Ermittlung von Zustandsklassen, welche dieses Petri-Netz im Laufe des Schaltens annehmen kann, erhält man zum Schluss ein Gleichungssystem für die Zeiten, welches mit Methoden aus der linearen Optimierung gelöst wird. Dabei können verschiedene Ergebnisse abgeleitet werden: Es kann die minimale und die maximale Schaltdauer der gesamten Sequenz berechnet werden und ob die vorhandenen Zeiteinschränkungen bei gegebener Schaltsequenz einhaltbar sind. Es können auch Zeitintervalle für die Transitionen gefunden werden, welche die Nichterreichbarkeit von bestimmten Zuständen hervorrufen. Wenn die Transitionssequenz in einer gegebenen Zeit nicht ausführbar ist, können die Zeitintervalle ermittelt werden, in welchen diese Transitionssequenz ausführbar wird.

6.3 Eigenschaften von Gefärbten Zeitintervall-Petri-Netzen

Im Folgenden werden einige Aspekte beschrieben, die bei den bekannten Petri-Netzeigenschaften Erreichbarkeit, Lebendigkeit, Konfliktfreiheit und Beschränktheit [Starke 90, Popo 89] auftreten, wenn man diese für die bereits beschriebenen Gefärbten Zeitintervall-Petri-Netze anwendet. Wenn die Schaltzeitpunkte von Transitionen innerhalb von Intervallen liegen müssen, können sich die konkreten Eigenschaften eines Gefärbten Petri-Netzes verändern.

Der **Erreichbarkeitsgraph** für ein Gefärbtes Zeitintervall-Petri-Netz kann aus den oben beschriebenen Zustandsklassen erstellt werden. Dabei wird ein Graph $RG_Z(z_{C0})$ als der Erreichbarkeitsgraph eines Gefärbten eingeschränkten Zeitintervall-Petri-Netzes bezeichnet, wenn seine Knoten die ganzzahligen Zustände sind und seine Bögen als Triple (z_C, t, z_C') und (z_C, τ, z_C') bezeichnet werden, wobei $z_C \xrightarrow{t} z_C'$ und $z_C \xrightarrow{\tau} z_C'$ sind.

Die entstandenen Zustandsklassen werden wie Zustände in herkömmlichen Erreichbarkeitsgraphen behandelt. Es ist allerdings möglich, dass dabei sehr große und eventuell auch unendliche Erreichbarkeitsgraphen entstehen. Lässt sich der Erreichbarkeitsgraph konstruieren, so lässt sich die **Erreichbarkeit** eines Zustandes wie bei den normalen Petri-Netzen in dem Erreichbarkeitsgraphen feststellen.

Die Definition der **Lebendigkeit** bleibt bei gefärbten Zeitintervall-Petri-Netzen ohne Veränderungen. Aber die Bedeutung dieser Definition wird etwas anders: Da ein gefärbtes Netz dadurch gekennzeichnet ist, dass mehrere Transitionen zu einer Transition zusammengefasst sind, kann es passieren, dass diese komplexe Transition nicht für alle Farben passierbar ist. Im Ursprungsnetz (vor der Faltung) entspricht diese Farbe der Transition einer nicht lebendigen Transitionen. Trotzdem ist im gefärbten Netz die zusammengefasste Transition lebendig. Dabei besteht die Möglichkeit, dass ein lebendiges Gefärbtes Petri-Netz in ein schwach lebendiges Netz entfaltet wird. Die Zeiteigenschaften kann man dadurch berücksichtigen, dass die Eigenschaft über den Erreichbarkeitsgraphen ermittelt wird, der wie oben beschrieben konstruiert wurde.

Die **Konfliktfreiheit** entspricht wie bei den anderen Eigenschaften von Gefärbten Zeitintervall-Petri-Netzen in der Definition der Konfliktfreiheit von normalen Petri-Netzen. Dabei wird diese in dynamische und statische untergliedert. Konflikte können dabei zwischen Transitionen, aber auch zwischen Farben in einer Transition auftreten. Die dynamischen Konflikte werden zeitabhängig. Durch die Intervalle kann es sein, dass Konflikte im Netz mit Zeitbe-

wertung nicht mehr auftreten, da die gleichzeitige Schaltfähigkeit durch die Intervalle nicht mehr existiert.

Auch die Definition der **Beschränktheit** ist gleich der Definition in normalen Netzen. Ein Netz ist **k-beschränkt**, wenn bei unendlicher Platzkapazität sich nicht mehr als k Marken gleicher Farbe in einem Platz befinden. Damit kann folgende Definition angewandt werden:

Ein gefärbtes Petri-Netz $CN=(P,T,F,C,V,K,m_0,\zeta)$ ist k-beschränkt, wenn für das Netz $(P,T,F,C,V,\{\infty,\dots,\infty\},m_0,\zeta)$ in jedem erreichbaren Zustand m gilt:

$$\forall p \in P, c \in C, n \in \mathbb{N}: (n,c) \in m(p) \Rightarrow n < k \text{ [Roth 2002].}$$

Bei Untersuchung dieser Eigenschaft im oben beschriebenen Erreichbarkeitsgraphen werden die Zeitbewertungen mit berücksichtigt.

In Kap. 5.3 wurden Einschränkungen der Gefärbten Petri-Netze benannt, die durch Transformation aus den betrachteten Ausgangsdiagrammen entstehen. Die Auswirkungen auf einfachere Analysealgorithmen bezüglich der Eigenschaften wurden nicht untersucht, da, wie eben beschrieben, die Methoden allgemeiner Gefärbter Zeitintervall-Petri-Netze anwendbar sind.

6.4 Toolrealisierung

Für die Arbeit wurde das Tool entwickelt. Die in diesem Tool realisierten Algorithmen erlauben die Überprüfung von Gefärbten Zeitintervall-Petri-Netzen auf ihre Eigenschaften wie Lebendigkeit, Konfliktfreiheit, Erreichbarkeit sowie die Ausführbarkeit von Transitionssequenzen. Das Tool hat ein Windows-ähnliches Interface. Um Petri-Netze aus unterschiedlichen Formaten einzulesen, sind in diesem Programm entsprechende Module integriert. Da die grafische Darstellung von in Textformat gespeicherten Petri-Netzen nicht vollständig lösbar ist und dieses Tool nur beispielhaft die Funktionen der oben beschriebenen Algorithmen nachweisen soll, wurde eine sehr einfache Form der Darstellung von Petri-Netzen gewählt (Abbildung 106).

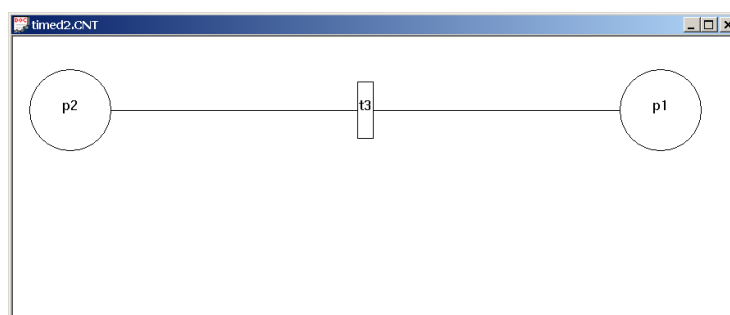


Abbildung 106: Darstellungsbereich vom Petri-Netz

In diesem Darstellungsbereich kann man recht einfach durch das Petri-Netz navigieren. Dabei gelangt man durch Anklicken von gezeigten Elementen in ein zu diesen Elementen gehöriges Eigenschaftsfenster (Abbildung 107 [Roth 2002])

Eigenschaften				
Transition t3				
Intervall	Zeit	Vorplätze	Nachplätze	
[2,2]	inaktiv	p2 (SW)	p1 (SW)	
[1,3]	inaktiv	p2 (SW)	p1 (SW)	

Abbildung 107: Eigenschaftsfenster

Durch die Anwendung des Analysemenüs können folgende Eigenschaften des Petri-Netzes überprüft werden:

1. Erreichbarkeit durch Erstellung eines Erreichbarkeitsgraphen (Abbildung 108 [Roth 2002]). Dabei kann auch die Erreichbarkeit eines Zustands bei Berücksichtigung von Markierungen und bereits vergangenen Zeiten für die Schaltfähigkeit der Transitionen berechnet werden.

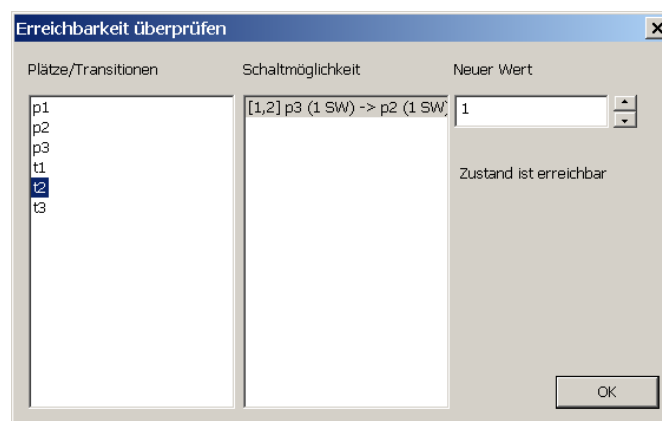


Abbildung 108: Überprüfung der Erreichbarkeit eines Zustandes

2. Konfliktfreiheit. Dabei wird das Netz sowohl auf statische als auch auf dynamische Konflikte überprüft.
3. Lebendigkeit. Sie kann für das gesamte Netz oder nur für ausgewählte Transitionen untersucht werden.
4. Transitionssequenzen. Dazu werden diese mit Hilfe eines speziellen Menüs eingegeben (Abbildung 109 [Roth 2002]).

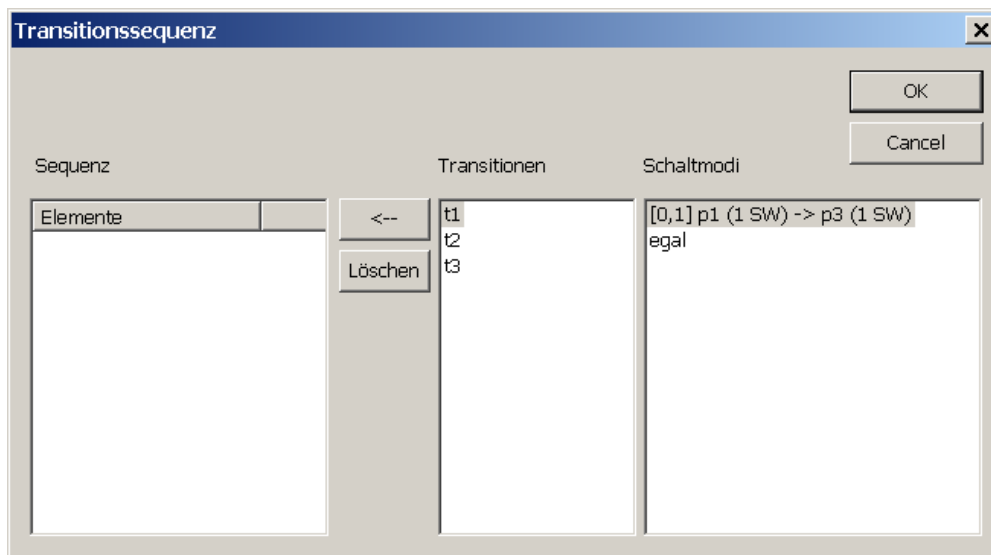


Abbildung 109: Eingabe einer Transitionssequenz

Die zu untersuchenden Transitionen werden ausgewählt und in eine Transitionssequenzliste übernommen. Dabei können auch die Schaltmöglichkeiten von Transitionen eingeschränkt werden. Bei der Übernahme von Transitionen mit mehreren Schaltmöglichkeiten werden alle Sequenzen berechnet. Die aufgestellte Liste der Transitionssequenz wird schrittweise durch die Berechnung von neuen Zustandsklassen abgearbeitet. Dabei weist eine Zustandsklasse, wie oben bereits beschrieben, die aktuelle Markierung und die Zeitbeschränkungen für die Variablen auf. Beim Auftreten von Transitionen mit mehreren Schaltmöglichkeiten werden alle diese Varianten berechnet, und später wird für alle Varianten die Optimierung durchgeführt.

Die Ergebnisse der Berechnung und der Optimierung aller Transitionssequenzen werden im Ergebnisfenster dargestellt. Dabei werden die maximale und die minimale Schaltdauer der Sequenz und die Zeitwerte der Transitionen, die zu diesen Ergebnissen geführt haben, gezeigt (Abbildung 110 [Roth 2002]).

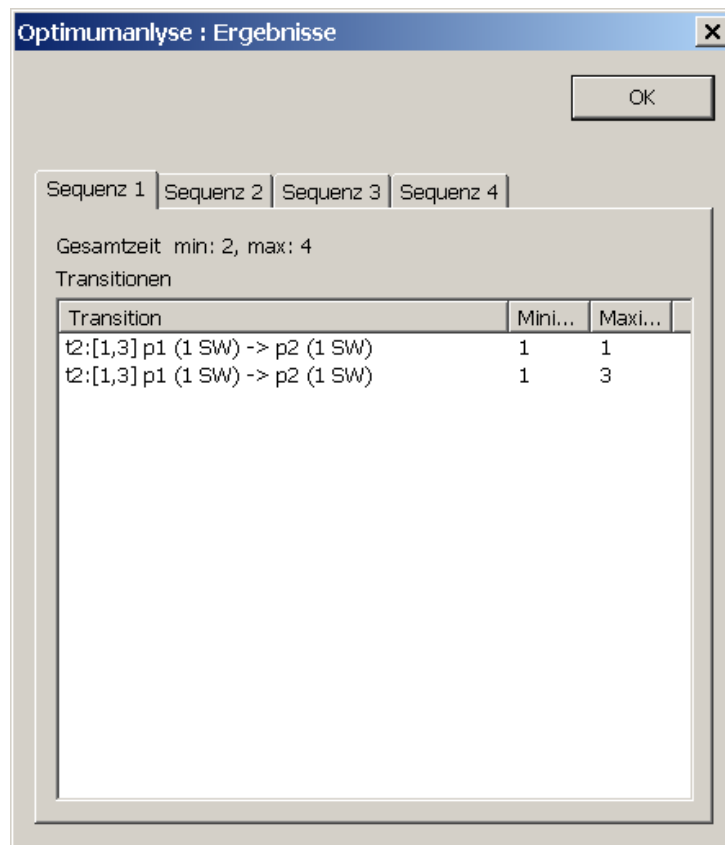


Abbildung 110: Ergebnisse der Berechnungen zur Transitionssequenz

Zusammenfassung und Ausblick

Die ständig steigende Komplexität und die Anforderungen an immer mehr Sicherheit von verteilten eingebetteten Rechnersystemen fordern die Einbeziehung von grafischen und objektorientierten Darstellungsmitteln sowie Werkzeugen in den Entwurf solcher Systeme. Diese ermöglichen einen effektiveren Entwurf und dessen parallele Überprüfung.

Ziel dieser Arbeit war die Entwicklung einer neuen Methode auf Basis erweiterter Diagramme der Unified Modelling Language (UML). Dabei sollte sich diese Methode beim Entwurf von eingebetteten verteilten Echtzeitsystemen auf die Spezifik dieser Systeme orientieren, welche in Heterogenität und Komplexität, Nebenläufigkeit von Prozessen, sowohl parallele als auch sequentieller Ausführung von Aktionen sowie Verteilung und Echtzeitbedingungen besteht. Die in dieser Arbeit vorgeschlagenen und beschriebenen gefärbten Zustands-, Aktivitäts- und Sequenzdiagramme erweitern deren Anwendungsbereich und bieten neue Modellierungsmöglichkeiten, vor allem in Form der Unterscheidung von Objekten durch Farben und der dabei entstehenden Kooperations- und Synchronisationsstrukturen. Dadurch werden komplexere Modelle möglich und die Übersichtlichkeit steigt deutlich gegenüber den in der UML vorhandenen Diagrammen. Das gilt insbesondere für sehr komplexe und verteilte Systeme, die eine Vielzahl weitgehend ähnlicher Komponenten enthalten. Die hier beschriebenen gefärbten Zustands-, Aktivitäts- und Sequenzdiagramme helfen, die grafischen Darstellungen für derartige Systeme übersichtlicher, kompakter und verständlicher zu gestalten.

Um die Zielstellung zu motivieren, erfolgte am Anfang der Arbeit eine Aufstellung über wesentliche Eigenschaften und Besonderheiten von verteilten eingebetteten Echtzeitsystemen und eine Darstellung der Problembereiche bei der Modellierung von solchen Systemen. Danach erfolgte eine kurze Beschreibung verschiedener, bei dem Entwurf solcher Systeme in Frage kommender Modellierungsansätze. Dabei war der Schwerpunkt die Darstellung von Konzepten der Objekttechnologie und der UML als Standard für die Objektorientierte Modellierung. Diese eröffnet eine Vielzahl von Möglichkeiten zum Entwurf von verschiedenen Sys-

temen, darunter auch von verteilten eingebetteten Echtzeitsystemen. Um eine Möglichkeit zu schaffen, bei dem Entwurf solcher Systeme auch formale Verifikationsansätze einzubeziehen, wurden hier auch Grundprinzipien von höheren Zeitintervall-Petri-Netzen erläutert. Diese eignen sich gut zur Formalisierung des Verhaltens verteilter eingebetteter Echtzeitsysteme mit dem Zweck der Verifikation mit bekannten und weiterentwickelten Verifikationsmethoden. Als Ergebnis der in diesem Kapitel durchgeführten Untersuchungen und der dabei festgestellten Mängel der vorhandenen Methoden wurde die Notwendigkeit zur Erweiterung von dynamischen objektorientierten Modellierungsansätzen für den Entwurf von verteilten eingebetteten Echtzeitsystemen abgeleitet. Die vorhandenen Diagramme und Methoden wurden weiter entwickelt. Durch die entwickelten gegenseitigen Transformationen werden sie zu einander passfähig. Damit wird der Entwurf von Systemen besser unterstützt und beherrschbarer.

Der erste Schwerpunkt der eigenen Arbeiten wurde durch die grundlegenden Definitionen für die Erweiterung von drei bei der dynamischen Modellierung besonders verbreiteten Diagrammtypen der UML gebildet: Message Sequence Charts (Sequenzdiagramme), State Charts (Zustandsdiagramme) und Activity Charts (Aktivitätsdiagramme), welche ausführlich in Kapitel 3 beschrieben sind. Da sich die vorgenommenen Erweiterungen auf die Eigenschaft „Farbe“ beziehen, wurden diese Diagrammtypen als gefärbte Diagramme bezeichnet. Das deutet auch auf die Übernahme von Teilen des etablierten Konzeptes der höheren Petri-Netze, speziell der Gefärbten Netze, auf die durch die UML weit verbreiteten Diagrammtypen hin. Hier wurden die Syntaxerweiterungen für diese Diagrammtypen sowie die Erläuterungen zur Funktionsweise vollständig angegeben und es wurde dabei gezeigt, wie durch Zusammenfassung von ähnlichen Abläufen in diesen Diagrammen die Parallelität von weitgehend identischen Teilprozessen in gleichen Strukturen beschrieben werden kann. Durch die Einführung von Farben in die Diagramme wurde weiterhin auch gezeigt, wie zusätzliche Informationen unter Beibehaltung der Übersichtlichkeit des Modells angebar sind. Die Ähnlichkeiten zum Farbkonzept von Petri-Netzen sind günstig für die spätere Umwandlung von Modellen der entwickelten Diagrammtypen ineinander über Gefärbte Petri-Netze. Als wichtige Komponente für die Modellierung und Analyse von Echtzeiteigenschaften wurde auch ein Zeitkonzept auf Basis von Zeitintervallen für die gefärbten Diagramme ausgearbeitet.

Im folgenden Kapitel wurden die Anwendbarkeit und die Vorteile der Modellierungstechnik mit gefärbten Aktivitäts-, Zustands- und Sequenzdiagrammen an einem konkreten, real existierenden Projekt gezeigt. Dabei ist klar zu sehen, dass schon bei kleineren Projekten die normalen UML-Diagramme ziemlich schnell zu groß und zu unübersichtlich werden. Der entwickelte Ausweg, der Einsatz der in dieser Arbeit geschaffenen gefärbten Diagramme, hilft die Anschaulichkeit der Modelle zu verbessern. Das ist bei dem beschriebenen Projekt gut erkennbar. Die redundanten Teile der Grafiken werden entfernt. Der Überblick über das gesamte Projekt wird besser und die Lesbarkeit der Diagramme wird erleichtert.

Als weiterer wichtiger Bestandteil dieser Arbeit sind die Algorithmen für die Transformation der beschriebenen Diagramme über Gefärbte Petri-Netze ineinander zu nennen. Diese wurden in Kapitel 5 dargestellt. Sie ermöglichen die effektive Arbeit an einem Projekt mit mehreren

Diagrammtypen, eventuell für verschiedene Sichten und Hierarchieebenen, indem Teilmodelle in verschiedene Diagramme konvertiert und kombiniert werden können. Weiterhin ist damit auch die Verifikation der Entwürfe, die mit den Ursprungsdiagrammen modelliert wurden, auf Basis der erzeugten Gefärbten Petri-Netze über die hier entwickelten Analysemethoden möglich. Als Ausgangspunkt für die Transformationen wurde am Anfang ein Vergleich von Zustandskonzept, Ereigniskonzept, Zustandsübergangskonstruktionen und von verschiedenen Entscheidungs-, Synchronisations- und Vereinigungskonstrukten für die gefärbten Zustands-, Aktivitäts- und Sequenzdiagramme durchgeführt. Dadurch wurde die Möglichkeit der Transformationen nachgewiesen. Die Transformations- und Rücktransformationswege wurden schrittweise beschrieben. Die Transformation von verschiedenen Elementen und Konstrukten wurde ausführlich in Tabellen zusammengefasst. Alle hier dargestellten Algorithmen werden beispielhaft durch verschiedene Tools, welche auf Sequenz-, Zustands- oder Aktivitätsdiagramme orientiert sind, unterstützt. Die Grundprinzipien der Funktionsweise dieser Tools wurden kurz zusammengefasst.

Weiterhin wurde eine zeitliche Verifikation von auf Zeitintervalle erweiterten gefärbten Diagrammen bearbeitet. Um die bei der Transformation aus den gefärbten Diagrammen gewonnenen Gefärbten Petri-Netze analysieren zu können, wurde die zeitliche Verifikationsmethode weiter entwickelt. Diese Erweiterung einer für normale Petri-Netze bekannten Methode schließt jetzt zusätzlich den Färbungsaspekt ein. Dabei wurde eine Definition von gefärbten Zeitintervall-Petri-Netzen angegeben und eine Verifikationsmethode für deren dynamische oder zeitbehafte Aspekte entwickelt, welche Aussagen über die Einhaltung von zeitlichen Restriktionen und den möglichen Nachweis von dynamischen Konflikten liefert. Anschließend wurde in diesem Kapitel die Funktionsweise des zur Durchführung der Verifikation entwickelten Tools dargestellt.

Die folgenden Arbeiten sollten sich mit der kompletten Unterstützung der Transformationen durch praktisch einsetzbare Tools beschäftigen, da die hier dargestellten Tools nur die Implementierbarkeit nachweisen. Dabei wäre es wünschenswert, ein gemeinsames Tool für die drei dynamischen Diagrammtypen zu schaffen, welches nicht nur die Algorithmen zur direkten Transformation in Gefärbte Petri-Netze, sondern auch die Rücktransformationen und die Transformationen von Diagrammen ineinander unter Einbeziehung der ausgearbeiteten Verifikationsalgorithmen unterstützt. Bei Vorhandensein eines geeigneten Petri-Netz-Tools mit grafischer Darstellungsmöglichkeit sollte das Transformationstool in einem Exportformat auch dessen Eingabeformat unterstützen. Gleiches gilt für die Verifikation auf Basis der Gefärbten Petri-Netze, deren Ergebnisse auf das Niveau der Ursprungsdiagramme zurück transformiert werden sollten. Die Realisierung von Simulatoren wäre auch sehr vorteilhaft für die bessere Untersuchung der Modelle. Weiterhin könnten auch Untersuchungen in Richtung der weiteren Optimierung der gefundenen Algorithmen und Lösungen geführt werden.

Auf theoretischem Gebiet sollten Arbeiten zur Entwicklung einer verallgemeinernden Gegenüberstellung von zustands- und ereignisorientierten Beschreibungsmitteln und deren hybriden Varianten erfolgen, die mehr als nur die unmittelbaren UML-Diagramme einschließen sollten.

Ergebnis könnte die Verträglichkeit und gegenseitige Ergänzung von Entwürfen, Methoden, Eigenschaften und Möglichkeiten der verschiedenen Modellierungsmittel und der mit ihnen verbundenen Verifikationsmethoden sein. Sinnvoll ist in diesem Zusammenhang auch die Betrachtung hierarchischer Konstrukte, die einen Systementwurf auf mehreren Abstraktionsebenen ermöglichen.

Auf dem Gebiet der Verifikation von Echtzeiteigenschaften sollten Untersuchungen durchgeführt werden, die es ermöglichen, Zeitanforderungen in frühen Entwurfsetappen zu definieren und bei der hierarchischen Verfeinerung im Laufe des Systementwurfes so zu behandeln, dass diese bei der Überführung in ein realisierendes Diagramm verifiziert werden können.

Literaturverzeichnis

- [Abel 87] Abel D.: Modellbildung und Analyse ereignisorientierter Systeme mit Petri-Netzen, VDI-Verlag GmbH, Düsseldorf, 1987
- [AbStMa 2002] Abstract State Machines. 2002. – <http://www.eecs.umich.edu/gasm/>
- [AlCoHe 93] Alur R., Courcoubetis C., Henzinger T.A., Ho P.-H.: Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossmann: Hybrid Systems I. Lecture Notes in Computer Science 736, S. 209-229, Springer-Verlag, 1993
- [AlDaBa 91] Alla H., David R., Le Bail, J.: Hybrid Petri nets. Proceedings of the European Control Conference, Grenoble, 1991.
- [Ambe 2001] Amberg C.: Entwicklung und Untersuchung von Gefärbten Sequenzdiagrammen, Diplomarbeit. TU Ilmenau, 2001
- [BaBö 93] Balke L., Boehling K. H.: Einführung in die Automatentheorie und Theorie formaler Sprachen. BI-Wiss.-Verl., 1993.
- [BaChGi 2000] Balarin F., Chiodo M., Giusto P., Hsieh H., Jurecska A., Lavagno L., Passerone C., Sangiovanni-Vincentelli A., Sentovich E., Suzuki K., Tabbara B.: Hardware-Software Co-Design of embedded Systems, Kluwer Academic Publishers, USA, 2000
- [Baru 2002] Barun S.: Toolmäßigen Implementierung von Modellierungsmethoden, Diplomarbeit, Ilmenau, 2002
- [Baum 90] Baumgarten B.: Petri-Netze: Grundlagen und Anwendungen, Mannheim [u.a.]: BI-Wiss.-Verl., 1990
- [BeGr 95] Best E., Grahlman B.: Programming Environment based on Petri nets. Documentation and User Guide Version 1.4 PEP. Universität Hildesheim, Institut für informatik, November 1995.
- [BeGr 98] Best E., Grahlman B.: Programming Environment based on Petri nets. Documentation and User Guide Version 1.8 PEP. University of Oldenburg, (Draft) December 1998.
- [Berg 2002] Berg A.: Modellierungsmethodik mit Gefärbten Zeitintervallzustands-Aktivitäts- Sequenzdiagrammen, Diplomarbeit, Ilmenau, 2002
- [Berm 01] Bermbach R.: Embedded Controller: Eine Einführung in Hard- und Software. Hanser Verlag, München, 2001, ISBN 3-446-19434-7
- [BiGu 2000] Bitsch, F., Gunzert M.: Formale Verifikation von Softwarespezifikationen in ASCET-SD und MATLAB. (2000). – Fachtagung Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, <http://www.ias.unistuttgart.de/de/forschung/pub/paper/va2000/paperbt.pdf>
- [BoMo 1979] Boyer R. S., Moore J. S.: A Computational Logic. Academic Press, New York, 1979
- [BoMo 1988] Boyer R. S., Moore J. S.: A Computational Logic Handbook. Academic Press, New York, 1988

- [BoRuJa 00] Booch G., Rumbaugh J., Jacobson I.: The Unified Modeling Language User Guide (Authorized translation from English Language Edition published by Addison Wesley Longman, Inc.) Translation by DMK Press, 2000 Verlag DMK, Moskau, 2000.
- [BrHoKr 97] Broy M., Hofmann C., Krüger I., Schmidt M.: A graphical description technique for communication in software architectures. Technical Report TUM-I9705, Technische Universität München, 1997.
- [BüLe 94] Büning H., Lettmann T.: Aussagenlogik: Deduktion und Algorithmen, Stuttgart : Teubner, 1994
- [BuMeRo 96] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M.: System of Patterns. Pattern-Oriented Software Architecture. Wiley, 1996.
- [Burk 97] Burkhardt R.: UML – Unified Modeling Language: Objektorientierte Modellierung für die Praxis, Bonn: Addison-Wesley-Longmann, 1997
- [CaDi 1993] Capellmann C., Dibold H.: Petri Net Based Specifications of Services in an Intelligent Network. Experiences Gained from a Test Case Application. In: Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993, Lecture Notes in Computer Science Vol. 691 (1993), S. 542–551
- [Cape 1998] Capellmann C., Christensen S., Herzog U.: Visualising the Behaviour of Intelligent Networks. In: Services and Visualization, Towards User-Friendly Design, Lecture Notes in Computer Science Vol. 1385 (1998), S. 174–189
- [Cape 1999] Capellmann C., Dibold H., Herzog U.: Using High-Level Petri Nets in the Field of Intelligent Networks. In: Application of Petri Nets to Communication Networks, Lecture Notes in Computer Science Vol. 1605 (1999), S. 1–36
- [Chav 1992] Chaves J.: Formal Methods at AT&T: An industrial usage report. In: Proceedings of Formal Description Techniques IV (1992), S. 83–90
- [Chou 99] Choikha M.: Entwurf diskret-kontinuierlicher Steuerungssysteme - Modellbildung, Analyse und Synthese mit hybriden Petrinetzen, Fortscjr.-Ber. VDI Reihe 8 Nr.797. Düsseldorf: VDI Verlag 1999
- [Clar 1993] Clarke E. M., Grumberg O., Hiraishi H., Jha S., Long D. E. ; McMillian, D. L. ; Ness, L. A.: Verification of the Futurebus+ cache coherence protocol. In: Proceedings of CHDL (1993)
- [Clau 98] Claussen U.: Objektorientiertes Programmieren. Mit Beispielen und Übungen in C++. Zweite, überarbeitete und erweiterte Auflage, Springer-Verlag Berlin Heidelberg, 1998.
- [ClEm 1981] Clarke E. M., Emerson E. A.: Synthesis of synchronisation skeletons for branching time temporal logic. In: Logic of Programs: Workshop, Lecture Notes in Computer Science Vol. 131 (1981)
- [ClGr 1999] Clarke E. M., Grumberg O., Peled D. A.: Model Checking. The MIT Press, Cambridge, Massachusetts, London, England, 1999
- [ClWi 1996] Clarke E. M., Wing J. M.: Formal methods: state of the art and future di-

- rections. In: ACM Computing Surveys 28 (1996), Nr. 4, S. 626–643. –
citeseer.nj.nec.com/clarke96formal.html
- [ClZh 1993] Clarke E., Zhao X.: Analytica: A theorem prover for Mathematica. In: The
Mathematica Journal (1993), S. 56–71
- [Corn 1995] Cornes C., Courant J., Filliatre J.-C., Huet G., Manoury P., Paulin-
Mohring C., Munoz C., Murthy C., Parent C., B.: The coq proof assistant
reference manual version 5.10. Technical Report 177 (July), INRIA. 1995.
– [http://pauillac.inria.fr/coq/systeme coq-eng.html](http://pauillac.inria.fr/coq/systeme%20coq-eng.html)
- [Cox] Cox K.: Statechart Diagrams, online im Internet: [http://dec.bourne-
mouth.ac.uk/staff/kcox/UMLStatecharts/index.htm](http://dec.bournemouth.ac.uk/staff/kcox/UMLStatecharts/index.htm)
- [CrKrMe 1988] Craigen D., Kromodimoelio S., Meisels I., Neilson A., Pase B., Saaltink
M.: m-EVES: A tool for verifying software. In: Proceedings of the 10th In-
ternational Conference on Software Engineering (1988), S. 324–333
- [DaAl 87] David R., Alla H.: Continuous Petri Nets. In: Proceedings of the European
Workshop on Applikations and Theory of Petri Nets, S. 275-294, Spanien,
1987
- [DaHa 99] Damm W., Harel D.: LSCs: Breathing Life into Message Sequence Charts.
In FMOODS'99 IFIP TC6/WG6.1 Third International Conference on For-
mal Methods for Open Object-Based Distributed Systems, 1999.
- [DaLa 92] David R., Alla H.: Petri Nets & Grafcet – Tools for modelling discrete
event systems. Prentice Hall, New York, 1992
- [DeKo 98] Demongodin I., Koussulas T.N.: Differential Petri Nets: Representing
Continuous Systems in a Discrete-Event World. IEEE Transaction on
Automatic Control, 43(4):573-578, 1998
- [DeSc 97] Decknatel G., Schnieder E.: modelling Railway Systems with Hybrid Petri
Nets, In: [Zaytoon 1998], S. 309-315
- [Dese 92] Desel J.: Struktur und Analyse von Free-Choice-Petrinetzen, Wiesbaden:
Dt. Univ.-Verl., 1992
- [DFG-AB 03] Duridanova V., Hummel T., Saffert E.: Entwurf eingebetteter paralleler
Steuerungssysteme für integrierte multi-axiale Antriebssysteme, DFG-
Abschlussbericht, 2003
- [DFG-ZB 01] Duridanova V., Hummel T., Saffert E.: Entwurf eingebetteter paralleler
Steuerungssysteme für integrierte multi-axiale Antriebssysteme, DFG-
Zwischenbericht, 2001
- [DiDrHu 1992] Dill D. L., Drexler A. J., Hu A. J., Yang C. H.: Protocol verification as a
hardware design aid. In: IEEE International Conference on Computer De-
signs: VLSI in Computers and Processors (1992), S. 522–525
- [Döri 00] Döring M.: Studienarbeit: Analyse, Modellierung und Verifikation von
hybriden Systemen am Beispiel eines Positionsmesstisches. Technische
Universität Ilmenau, 2000.
- [Döri 02] Döring M.: Verifikation Von Intervall Petri-Netzen, Technische Universi-
tät Ilmenau, Diplomarbeit, 2002

- [Doug 2000] Douglass P. P.: Doing Hard Time: Developing Real-Time systems with UML, objects, frameworks, and patterns. Addison Wesley, 2000.
- [Doug 98] Douglass P. P.: Real-Time UML: Developing efficient objects for embedded systems. Addison-Wesley, 1998.
- [Drath 99] Drath R.: Modellierung hybrider Systeme auf der Basis modifizierter Petri-Netze. Dissertation, TU Ilmenau, 1999.
- [DuHu 01] Duridanova V., Hummel T.: Modelling of Embedded Mechatronic Systems Using Hybrid Petri Nets.in: M. H. Hamza (Ed.): IASTED International Conference Modelling, Identification, and Control, Feb. 19-22, 2001, Innsbruck, Austria, IASTED/ACTA Press 2001, vol. 1, pp. 521-526. ISBN 0-8898-6316-4
- [DuHuFeFe 04] Duridanova V., Hummel T., Fengler O., Fengler W.: Verifikation von Spezifikationsmodellen mit Intervall-Petri-Netzen, Kaiserslautern 2004
- [Dumk] Dumke R.: UML – Tutorial. online im Internet: <http://ivs.cs.uni-magdeburg.de/~dumke/UML/>
- [Dunk] Dunkel J.: Einführung in die Unified Modelling Language, Fachhochschule Hannover, Fachbereich Informatik, online im Internet: <http://java.rrzn.uni-hannover.de/jug/uml/>
- [FeFeDu 02a] Fengler O., Fengler W., Duridanova V.: Colored Sequence Diagrams for Modeling Cooperating Processes, Proceedings of the IASTED International Conference on Modelling, Identification, and Control, held February 18&21, 2002, Austria
- [FeFeDu 02b] Fengler O., Fengler W., Duridanova V.: Modeling of complex automation systems using colored state charts, ICRA 2002: Proceedings of the 2002 IEEE International Conference on Robotics and Automation.May 11-15, 2002, Washington D.C. ISBN: 0-7803-7273-5 IEEE (c) 2002 #02CH37292C vol. 2, pp. 1901-1906
- [FeFeDu 02c] Fengler O., Fengler W., Duridanova V.: Transformation zwischen zustandsorientierten Beschreibungsmitteln der Elektronik und Informatik. Deutsche Forschungsgemeinschaft: Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen. WILEY-VCH&Co.KGaA, 2002, S.229-245
- [FeFeDu 02d] Fengler O., Fengler W., Duridanova V.: Extending the Modeling Efficiency of the UML Activity Diagram for the Design of Distributed Systems. Innovative internet computing systems: second international workshop; proceedings / IICS 2002, Kühlungsborn, Germany, June 20-22, 2002. - Berlin: Springer, 2002, S. 51-62.
- [FeHuFe 02] Fengler O., Hummel T., Wolfgang Fengler: Modellierung kooperierender Prozesse mit gefärbten Sequenzdiagrammen. 5. GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen. Tübingen, 25.-27.02.2002.in: J. Ruf (Hrsg.): 5. GI/ITG/GMM-Workshop: Methoden und Beschreibungsspra-

- chen zur Modellierung und Verifikation von Schaltungen und Systemen. Shaker-Verlag Aachen 2002, S. 199-208. ISBN 3-8265-9859-8
- [FeFeHu 04] Fengler O., Wolfgang Fengler, Hummel T.: Verification Method for Modelling Cooperating Processes with Coloured Sequence Diagrams, Proceedings of the IASTED International Conference on Modelling, Identification, and Control, held February 23-25, 2004, Schweiz
- [Feng 88] Fengler W.: Technische Applikation von Petrinetzen, Petri-Netze in Technik und Wirtschaft, Technische Universität Ilmenau, Institut für Technische und Theoretische Informatik, Fachgebiet Rechnerarchitektur, 88
- [FePh 91] Fengler W., Phillippow I.: Entwurf industrieller Mikrocomputersysteme. Carl Hanser Verlag, München Wien, 1991.
- [FiKr 1999] Figueiredo J.C.A., Kristensen L.M.: Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols. In: Proceedings of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/ CPN (1999), S. 21-40
- [Flau 97] Flaus J.M.: Hybrid Flow Nets For Batch Process Modelling and Simulation. In: Troch I. Und Breitenacker F., MATHMOD'97, S. 213-216, Vienna
- [FoMe 2002] Formal Methods Europe. 2002. – <http://www.fmeurope.org/>
- [Fric 96] Fricke O.: Statechart-orientierte Modellierung mit höheren Petri-Netzen. Kick-Off-Workshop der DFG-Forschergruppe PETRINETZ-TECHNOLOGIE, Berlin, 15. Juli 1996. Technische Universität Berlin, Institut für Kommunikations- und Softwaretechnik.
- [GaVaNaGo 94] Gajski D., Vahid F., Narayan S., Gong J.: Specification and Design of Embedded Systems; PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994
- [GeGoWe 99] Gehrke T., Goltz U, Wehrheim H.: Zur semantischen Analyse der dynamischen Modelle von UML mit Petri-Netzen. In: Schnieder (Hrsg.): Tagungsband der 6. Fachtagung Entwicklung und Betrieb komplexer Automatisierungssysteme (EKA '99), S. 547-566, Beyrich, Braunschweig 1999. Abstract. Paper (80 kB, 20pp).
- [Glas 2002] Glazounov A.: Verifikation von gefärbten Zeitintervallzustands-, Aktivitäts-, Sequenzdiagrammen, Diplomarbeit, Ilmenau, 2002
- [GoBi 2000] Gordon S., Billington J.: Modelling and Initial Analysis of the Resource Reservation Protocol Using Coloured Petri Nets. In: Proceedings of the Workshop on Practical Use of High-level Petri Nets (2000), S. 91-110
- [Gord 1987] Gordon M.: HOL: A proof generating system for higher-order logic. In: VLSI Specification, Verification and Synthesis (1987)
- [Gren 2003] Grenzdörfer M.: Modellierung eines Meßsystems auf Basis der UML, Hauptseminar, Ilmenau, 2003
- [GrGrRu 93] Grabowski J., Graubmann P., Rudolph E., "Towards a Petri net based semantics definition for message sequence charts," in O. Frgemand and A.

- Sarma (Eds.), *SDL'93: Using Objects*, Proc. 6th SDL Forum, North-Holland, 1993, pp. 179--190.
- [GrKrSt 99] Grosu R., Krüger I., Stauner T.: *Hybrid Sequence Charts*. Technical Report TUM-I9914, Technische Universität München, 1999.
- [GrWa 98] Grimm C, Waldschmidt K.: *Spezifikation hybrider Systeme*. In: Müller, W., Rammig, F. J. (Hrsg.): *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen - GI/ITG/GMM-Workshop, Paderborn, 1998*.
- [HaPo 98] Harel D, Politi M.: *Modelling Reactive Systems with Statecharts. The Statemate approach*. Printed and bound by R. R. Donnelley & Sons Company, 1998.
- [Henz 96] Henzinger T.A.: *The Theory of Hybrid Automata*. In *Proceedings of the 11 Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, Seiten 278-292, 1996
- [HePo 97] Heiner M., Popova-Zeugmann L.: *Worst-case Analysis of Concurrent Systems with Duration Interval Petri Nets (new version)*, Informatik- Berichte der HUB Nr. 83, 1997.
- [HePo 97] Heiner M., Popova-Zeugmann L.: *Worst-case Analysis of Concurrent Systems with Duration Interval Petri Nets*, Fachtagung EKA '97, Braunschweig, May 1997
- [HeRu 98] Henzinger T.A., Rusu V.: *Reachability verification for hybrid automata*. In: *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control (HSCC 1998)*. Lecture Notes in Computer Science 1386, S. 190-204, Springer-Verlag, 1998
- [HiKa 99] Hitz M., Kappel G.: *UML&Work: von der Analyse zur Realisierung*. Heidelberg: dpunkt-Verl., ISBN 3-932588-38-X, 1999
- [Hind 98] Hindel B.: *Technologie-Forum: Embedded Software 98*. Erlangen, Germany, 1998
- [Hoar 1969] Hoare C. A. R.: *An Axiomatic Basis for Computer Programming*. In: *Communication of the ACM 12 (1969)*, S. 576–580
- [HuJe 90] Hubert P., Jensen K., Shapiro R.M.: *Hierarchies in Coloured Petri Nets*. In: *Advances in Petri Nets 1990*. Lecture Notes in Computer Science 483, Springer Verlag, 1991
- [ITU-TS 96] ITU-TS: *Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, 1996.
- [ITU-TS 98] ITU-TS : *Recommendation Z.120 : Annex B*. Geneva, 1998.
- [ITU-TS 99] ITU-TS: *Recommendation Z.100 (11/99): Specification and Description Language (SDL)*, 1999
- [ITU-TS] ITU-TS : *Recommendation Z.120 (11/99) : MSC 2000*. Geneva, 1999
- [Jens 92] Jensen K.: *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use Volume 1*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992

- [Jens 95] Jensen K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use Volume 2. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1995
- [JeRo 91] Jensen D., Rozenberg G.: High-level Petri Nets – Theory and Application. Springer-Verlag, Berlin, 1991
- [John 1979] Johnson S. C.: Yacc: Yet Another Compiler Compiler. In: UNIX Programmer's Manual Bd. 2. New York, NY, USA : Holt, Rinehart, and Winston, 1979, S. 353–387. – URL cite-seer.nj.nec.com/johnson79yacc.html
- [KaLoNo 76] Kausmann U., Lommatzsch K., Nozicka F.: Lineare parametrische Optimierung : mit 47 Tabellen, Berlin : Akad.-Verl., 1976
- [KaMo 1995] Kaufmann M., Moore J. S.: ALC2: A Computation Logic for Applicative Common Lisp, The User Manual (Version 1.8). 1995. – <ftp://ftp.cli.com/pub/acl2/v1-8/acl2-sources/doc/HTML/acl2-doc.html>
- [KePf 90] Kempel Y., Pfander G.: Praxis der objektorientierten Programmierung. Carl Hanser Verlag, München, 1990
- [KePn 92] Kesten Y., Pnueli A.: Timed and Hybrid Statecharts and Their Textual Representation. In J. Nytopil (Hrsg.): Format Techniques in Real-Time and Fault-Tolerant Systems. Volume 571 of Lecture Notes in Computer Sciences, 1992
- [KIHo 00] Kluge O., Hommel G.: Message Sequence Chart Specifications with Time and their Representation as Stochastic Petri Nets. (Extended Abstract), In: Proceedings of the High Performance Computing Conference 2000 (HPC'2000), Washington DC, USA, April 2000.
- [KIPaEh 00] Kluge O., Padberg J., Ehrig H.: Modeling Train Control Systems: From Message Sequence Charts to Petri Nets. In: FORMS 2000 – Formale Techniken für die Eisenbahnsicherung, Fortsch.-Ber. VDI Reihe 12 Nr. 441. pages 25-42. Düsseldorf: VDI Verlag
- [Klug 02] Kluge O.: Petri Nets as a Semantic Model for Message Sequence Chart Specifications. In: Proceedings of INT 2002 : Second International Workshop on Integration of Specification Techniques for Applications in Engineering, Grenoble, France, April 2002.
- [Knor 95] Knorr R.: Spezifikation, Verifikation, Leistungsbewertung und Implementierung von Kommunikationsprotokollen mit hierarchischen High-Level-Netzen, Aachen: Shaker, 1995
- [KöQu 88] König R., Quaeck L.: Petri-Netze in der Steuerungstechnik, Berlin : Verl. Technik, c 1988
- [KöQu 89] König R. und Quäck L.: Petri-Netze in der Steuerungs- und Digitaltechnik, R.Oldenbourg Verlag, München 1989.
- [KrChJe 98] Kristensen L.M., Christensen S., Jensen K.: The practitioner's guide to coloured Petri nets. CPN Group, Department of Computer Science, University of Aarhus, Denmark, Springer-Verlag, 1998.

- [Krüg 2000] Krüger I. H.: Distributed System Design with Message Sequence Charts. Dissertation an der Technischen Universität München, Fakultät für Informatik, 2000
- [Lang 2000] Langbein M.: Message Sequence Charts für die formale Spezifikation von Embedded Systems. Diplomarbeit an der Technische Universität Ilmenau, 2000.
- [LeAr 98] Leinecker R., Archer T.: Die Visual C++ 6 Bibel. Mitp-Verlag, 1998.
- [Lesk 1975] Lesk M.E.: Lex - a lexical analyzer generator / AT&T Bell Laboratories. Murray Hill, N.J., 1975 (39). – Computing Science Technical Report 39
- [Lich 2004] Licht T.: Ein Verfahren zur zeitlichen Analyse von UML-Modellen beim Entwurf von Automatisierungssystemen, Dissertation. Ilmenau, 2004
- [LuPo 1992] Luo Z., Pollack R.: LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211 (May). Computer Science Dept., University of Edinburgh, 1992
- [LyMa 99] Lyngso R.B., Mailund T.: Textual Interchange Format for High-level Petri Nets. Computer Science Department University of Aarhus, Denmark
- [LyMa] Lyngso R., Mailund T.: Textual Interchange Format for High-level Petri Nets, Computer Science Department University of Aarhus DK-8000 Aarhus C, Denmark
- [MaRe 94] Mauw S., Reniers M. A.: An Algebraic Semantics of Basic Message Sequence Charts. The Computer Journal, 37(4), 1994.
- [MaRe 96] Mauw S., Reniers M. A.: Refinement in Interworkings. In: U. Montanari and V. Sassone, editors, CONCUR'96, volume 1119 of LNCS, pages 671-686. Springer, 1996.
- [Melz 98] Melzer S.: Verifikation verteilter Systeme mittels linearer- und Constraint Programmierung, Erschienen: München: Utz, Wiss., 1998
- [Merl 74] Merlin P., A: Study of the Recoverability of Computer Systems, Thesis, Department of Computer Science, University of California, Irvine, 1974
- [Merl 76] Merlin P., Faber D.J.: Recoverability of Communication Protocols, IEEE Transactions on Communications, vol. Com-24, No. 9, September 1976
- [MeTi 00] Meschkow A., Tikhomirov J.: Visual C++ und MFC. BHW-Sankt-Petersburg, 2000
- [Meye 90] Meyer B.: Objektorientierte Softwareentwicklung. Carl Hanser Verlag, München, 1990, ISBN: 3-446-15773-5
- [MoChDe 98] Moncelet G, Christensen S., Demmou H., Paludetto M., Porras J.: Analysing a mechatronic system with coloured Petri nets. Springer-Verlag, 1998.
- [Mühl 98] Mühlpfordt A.: Dissertation: Objektorientierte Geschäftsprozessmodellierung auf der Basis der UML, Ilmenau, 1998.
- [NüFeBö 99] Nützel J., Fengler W., Böhme T.: Objektorientierter Entwurf verteilter eingebetteter Echtzeitsysteme auf Basis höherer Petri-Netze. 6. Fachtagung "Entwurf Komplexer Automatisierungssysteme" (EKA'99), Braunschweig, Mai 1999

- [Nütz 99] Nützel J.: Objektorientierter Entwurf verteilter eingebetteter Echtzeitsysteme auf Basis höherer Petri-Netze, Ilmenau, Techn. Univ., Diss., 1999
- [OcPr 95] Ochsenschläger P., Prinoth R.: Modellierung verteilter Systeme : Konzeption, formale Spezifikation und Verifikation mit Produktnetzen, Braunschweig [u.a.]: Vieweg, 1995
- [Oest 98] Oestereich B.: Objektorientierte Softwareentwicklung; Analyse und Design mit der UML. Oldenbourg, Verlag München/Wien, 1998
- [OMG 2000] OMG: Unified Modeling Language Specification. Version 1.3, OMG (<http://www.omg.com>), 2000.
- [OMG 2004] OMG: Unified Modeling Language Specification. Version 1.5, OMG (<http://www.omg.com>), 2004.
- [OpRe 2002] Operations Research Lab. 2002. – <http://orlab.snu.ac.kr/>
- [Orek 00] Orekhov E.: Konverter der OTW-Zustandsdiagramme in höhere Petrinetze des PEP-Tools, 2000.
- [PeNeWo 2002] Petri Nets World. 2002. – <http://www.daimi.au.dk/PetriNets/>
- [Petr 1] Coloured Petri Nets at the University of Aarhus, online im Internet: <http://www.daimi.aau.dk/CPnets/>
- [Petr 62] Petri C. A.: Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962.
- [Popo 89] Popova-Zeugmann L.: Zeit-Petri-Netze – Diss, Humboldt-Universität Berlin 1989
- [Popo 94] Popova-Zeugmann L.: On Time Invariance in Time Petri Nets (Part I), Workshop Concurrency, Specification & Programming, Informatik-Berichte der HUB Nr. 36, 1994.
- [Popo 94] Popova-Zeugmann L.: On Time Invariance in Time Petri Nets (Part II), Workshop Concurrency, Specification & Programming, Informatik-Berichte der HUB Nr. 39, 1994.
- [Popo 97] Popova-Zeugmann L.: On Parametrical Sequences in Time Petri Nets, Workshop Concurrency, Specification & Programming, Warschau, Tagungsband, 1997.
- [Popo 97] Popova-Zeugmann L.: Strukturelle Untersuchungen von Schaltsequenzen in zeitabhängigen Petri-Netzen, 4. Workshop Algorithmen und Werkzeuge für Petrinetze, Berlin, Informatik-Bericht Nr. 85, Oktober 1997.
- [Popo 98] Popova-Zeugmann L.: Essential States in Time Petri Nets, Informatikberichte der HUB Nr. 96, 1998 .
- [Popo 99] Popova-Zeugmann L., Schlatter D.: Analyzing Path in Time Petri Nets, Fundamenta Informaticae (FI), 37, IOS-Press. 1999
- [Quäc 91] Quäc L.: Aspekte der Modellierung und Realisierung der Steuerung technologischer Prozesse mit Petri-Netzen. Teil 1. In: Automatisierungstechnik - at, Vol. 39, No. 4, pages 116-120. April 1991.
- [QuSi 1982] Queille J., Sifakis J.: Specification and verification of concurrent systems in CÆSAR. In: Proceedings of Fifth ISP (1982)

- [Ramc 76] Ramchandani C.: Analysis of asynchronous concurrent systems using timed Petri nets. Research Report MAC-TR 120, Massachusetts Institute of Technology, Feb. 1976.
- [Rang 2000] Rang C.: Studienarbeit:, Transformation eines Zustandsdiagramms in ein Aktivitätsdiagramms mit Hilfe von Petrinetzen, 2000
- [Rang 2001] Rang C.: Transformation von Zustands-, Aktivitäts-, Sequenzdiagrammen ineinander über eingeschränkte höhere Petri-Netze, Diplomarbeit, TU Ilmenau, 2001
- [Rati] Rational Software Corporation: Unified Modeling Language; Online im Internet: URL: <http://www.rational.com/uml>
- [Raut 96] Rautenberg W.: Einführung in die mathematische Logik: ein Lehrbuch mit Berücksichtigung der Logikprogrammierung, Braunschweig [u.a.]: Vieweg, 1996
- [Reis 85] Reisig W.: Systementwurf mit Netzen, Berlin, Heidelberg, New York, Tokyo: Springer, 85
- [Remp 99] Remp M.: Entwicklung und Verifikation von Steuerungssystemen auf der Basis hierarchischer Zustandsmodelle, Bochum: Ruhr-Univ., Lehrstuhl für Regelungssysteme und Steuerungstechnik, 1999
- [Rhap 99] Rhapsody ®2.1 Reference Guide, 1999.
- [Riab 2001] Riabov Y.: Entwicklung und Untersuchung von gefärbten Zeitintervall-Aktivitätsdiagrammen. Diplomarbeit an der Technische Universität Ilmenau, 2001.
- [Riab 2003] Riabov Y.: Transformation von gefärbten Verhaltensdiagrammen ineinander mittels höherer Petri-Netze, Hauptseminar, Ilmenau 2003
- [Roko 99] Pokosij E.: Toward verification of concurrent properties of time Petri nets, Siberian Division of the Russian Academy of Sciences A. P. Ershov Institute of Informatics Systems, 1999
- [Roku 92] Rokyta P.: Theoretische Grundlagen für Tools für Gefärbte Petri-Netze,
- [RoSt 2000] Roch S., Starke P. H.: INA - Integrated Net Analyzer Version 2.2 Manual. 2000. – <http://www.informatik.hu-berlin.de/~starke/ina.html>
- [Roth 2002] Roth J.: Erweiterung und Implementation von Verifikationsmethoden für Gefärbte Zeitintervall-Petrinetze, Diplomarbeit, Ilmenau, 2002
- [RoWi 91] Rosenstengel B., Winand U.: Petri-Netze: Eine anwendungsorientierte Einführung. 4., verbesserte und erweiterte Auflage, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1991.
- [SaSc 96] Saffert E., Schäffel C., Kallenbach E.: Control of an Integrated Multi-coordinate Drive. *Mechatronics'96*, 18.-20.09.1996, Guimaraes, Portugal, Proceedings Vol. 1, S. 151-156.
- [Schä 98] Schäffel C.: Untersuchung zur Gestaltung integrierter Mehrkoordinatnantriebe. ISLE Verlag 3-932633-03-2
- [Schm 99] Schmitt F.: Embedded-Control-Architekturen, Hanser. München, 1999

- [Schn 93] Schnieder E.: Prozessinformatik: Automatisierung mit Rechensystemen; Einführung mit Petrinetzen. 2., erweiterte Auflage, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1993.
- [Schu 99] Schubert S.: Konsistenz und Werkzeugunterstützung von Systemspezifikationen in UML, Diplomarbeit an der Universität Hannover, 1999.
- [ScWeWe 99] Scmitt F. J., Wendorff W. C., Westerholz K.: Embedded-Control-Architekturen: Desing für Embedded-Performance. München, Wien: Hanser, 1999, ISBN 3-446-19573-4
- [SeRu 98] Selic B., Rumbaugh J.: Die Verwendung der UML für die Modellierung komplexer Echtzeitsysteme. In: OBJEKTSpectrum 4, S.24-35, 1998.
- [SeRu 99] Selic B., Rumbaugh J.: Using UML for Modelling Complex Real Time System Architectures. A White paper, <http://www.rational.com/media/whitepapers/umlrt.pdf>
- [Slav 2001] Slavianov A: Entwicklung und Untersuchung von gefärbten Zustandsdiagrammen, Diplomarbeit, TU Ilmenau, 2001
- [Slav 2003] Slavianov A.:Transformation von gefärbten Verhaltensdiagrammen ineinander mittels höherer Petri-Netze, Hauptseminar, Ilmenau 2003
- [SIKr 89] Sloman M., Kramer J.: Verteilte Systeme und Rechnernetze, München [u.a.]: Hanser [u.a.], 1989
- [Sonn 92] Sonnenschein M.: Petri-Netze: Eine einführende Übersicht für Studierende der Informatik, Zweite, überarbeitete Auflage, Aachen, 1992.
- [Star 78] Starke M.: Petri-Netze. Herausgegeben vom Zentralinstitut für Kibernetik und Informationsprozesse der Akademie der Wissenschaften der DDR, Berlin, Juni 1978.
- [Star 90] Starke P. H.: Analyse von Petri-Netz-Modellen, Stuttgart, 1990
- [Stör 99] Störrle H.: *A Petri-net semantics für Sequence Diagrams*“, Ludwig-Maximilians-Universität München, München, 1999.
- [Stro 00] Stroustrup B.: Die C++ Programmiersprache. 4., aktualisierte und erweiterte Auflage. Deutsche Übersetzung der „Special Edition“ von Nicolai Josuttis und Achim Lörke. Pearson Education Deutschland GmbH, München, 2000.
- [Tane 95] Tannenbaum A.S.: Verteilte Betriebssysteme. München; New York; London, Toronto: Prentice-Hall-Verl., 1995, ISBN 3-930436-23-X
- [Teic 97] Teich J.: Digitale Hardware/Software-Systeme: Syntese und Optimierung, Berlin:Springer,1997
- [UML 1] UML Resource Page; Online im Internet;URL: <http://www.omg.org/uml/>
- [UML 2] OMG Unified Modeling Language Specification, Version 1.3, June 1999
- [UML 3] UML auf gut Deutsch, online im Internet: http://www.sigs.de/publications/docs/obsp/uml_d.html
- [UML 4] Glossar für das Themengebiet UML, online im Internet: <http://www.oose.de/oep/desc/glo-all.htm>

- [UML 6] Softwareentwicklung mit der Unified Modeling Language: online im Internet: <http://www-informatik.fh-harz.de/Grafiklabor/Software/uml/index.htm>
- [UML 7] Nestler M.: UML im Kontext: online im Internet:<http://www.inf.tu-dresden.de/~mn2/UML/Vortrag%20UML.htm>
- [UML] UML im Kontext: online im Internet: <http://www.inf.tu-dresden.de/~mn2/UML/Vortrag%20UML.htm>
- [VeTeReGr 2000] Verification & Testing Research Group. 2000. – <http://www.dcs.shef.ac.uk/research/groups/vt/mission.html>
- [Wiet98] Wieting R.: Modellbildung und Simulation mit Hybriden höheren Netzen. Dissertation, ISBN 3-8265-3291-0, Shaker Verlag Aachen, 1998
- [XuKu 98] Xu J., Kuusela J: Analyzing the execution architecture of mobile phone software with colored Petri nets. Nokia Research Center, Springer-Verlag, 1998.
- [YoSc 93] Yoneda T., Shibayama A., Schlingloff B.H., Clarke E.M.: Efficient verification of parallel real-time systems, Lect. Notes Comput. Sci., 1993
- [Zimm 97] Zimmermann A.: Modellierung und Bewertung von Fertigungssystemen mit Petri-Netzen, Technische Universität Berlin, Dissertation, 1997
Petri, C. A.: Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962. TU Ilmenau, 1992