

Komponentenbasierte Überwachung hybrider Systeme durch den Einsatz formaler Methoden

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Dipl.-Ing. Leila Mekacher
Aus Köln

Mannheim, 2016

Dekan: Professor Dr. Heinz Jürgen Müller, Universität Mannheim
Referent: Professor Dr. Colin Atkinson, Universität Mannheim
Korreferent: Professor Dr. Barbara Paech, Universität Heidelberg

Tag der mündlichen Prüfung: 17. November 2016

Danksagung

Ein besonderer Dank ergeht an Herrn Prof. Dr. Colin Atkinson, Leiter des Lehrstuhls für Softwaretechnik der Universität Mannheim, für die freundliche Betreuung meiner Doktorarbeit und für seine große Diskussionsbereitschaft im Rahmen der gemeinsamen Publikationen. Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin am Lehrstuhl für Automation der Universität Heidelberg. Für das gute Arbeitsklima und die Hilfsbereitschaft danke ich allen Mitarbeitern des Lehrstuhls. Ebenso danke ich allen Studenten, die im Rahmen von Diplomarbeiten und als wissenschaftliche Hilfskräfte unter meiner Anleitung Beiträge zu meiner Arbeit geleistet haben.

Weiterhin danke ich herzlich Herrn Dipl.-Ing. Frank Stolzenberger, der mir stets seine uneingeschränkte Unterstützung in den Bereichen Rechnereinsatz und Organisation zukommen ließ.

Ein ganz besonderes Dankeschön ergeht an Prof. Dr. Heinz Jürgen Müller, Dekan der Fakultät für Wirtschaftsinformatik und Wirtschaftsmathematik der Universität Mannheim, für seine moralische und praktische Unterstützung bei schwierigen Situationen, die das Abschließen dieser Arbeit verzögert hätten.

Nicht zuletzt möchte ich mich bei meiner Familie und meinen Freunden bedanken, die mir immer ein großer Rückhalt waren. Ein besonderer herzlicher Dank ergeht an meine Freundin Çigdem Yiler, die mir stets ihre uneingeschränkte moralische Unterstützung zukommen ließ und in den schwierigen Zeiten immer für mich da war. Für die Engelsgeduld und das entgegengebrachte Verständnis arbeitsintensiver Tage und Wochenenden, danke ich dabei besonders meinen beiden Söhnen Aziz und Ilyès. Ganz besonders herzlich danke ich meinen Eltern Jamila und Amor sowie meinem Bruder Kais. Sie haben mir nicht nur während des Studiums ihre uneingeschränkte Unterstützung zukommen lassen, sondern auch immer an mich geglaubt. Ihnen und meinen beiden Söhnen widme ich diese Arbeit.

Abstract

This thesis deals with the development of a new method for seamless component design and system monitoring based on a unified dependability model suitable for developing complex dynamic systems. It therefore advances the state-of-the-art in designing dependable systems. The underlying methods used for this purpose is the component-based design methodology KobrA since it support the systematic analysis and decomposition of components at different levels of abstraction and granularity. It also supports both “top-down” as well as “bottom-up” approaches that are suitable for developing efficient prototype system realizations.

To provide a formal real-time monitoring and fault detection mechanism, in this thesis the KobrA-method is enhanced with a formal modeling language designed to be understandable to both software developers and system engineers. The desired language should not only have a unique and strictly defined semantics it should support a unified description of the system components and monitoring components that allows the systematic integration of the monitoring over the entire development process as well as its execution during operation.

The underling formalism is based on Petri nets – powerful graph theoretical models that have been used in many different areas of software development for many years. They allow the description of component behavior and the conditions for the data flow through a network of nodes. The main advantages of Petri nets are firstly their formal mathematical formulation based on a solid theoretical foundation, and secondly, their ability to explicitly illustrate processes via a labeling concept. Petri nets also allow the representation of sequential and concurrent activities, the modeling and visualization of system behavior, and the synchronization of cooperative processes.

In this thesis the behavioral description of the monitoring components is achieved through a new class of Petri nets called “Modified Particle Petri Nets – MPPN”. This new class includes hybrid Petri nets for modeling the hybrid system behavior and a probabilistic “particle filter” extension to allow the monitoring process to be treated as a tracking problem. Petri nets not only support the complete and consistent description, graphical representation and simulation of processes it also support animation-based testing as early as the design phase. The combination of KobrA’s description formalism and Petri nets supports the intuitive, modular and hierarchical modeling of component-based systems in a formal way.

Supporting tools developed as part of this work allow the realization of the monitoring component to be generated directly from the specification. For this purpose, the Petri net

model is transformed into a compact, textual XML interchange format (eng., "Extensible Markup Language"), which is based on the PNML standard (eng., "Petri Net Markup Language"). This generic template includes the components' behaviors and the necessary parameters for the monitoring process.

There are two main motivations for the use of a single formal method, Petri nets, for both the specification and the implementation of component-based systems. The primary objective is for the system design to be described using a unified, clear language supporting the representation of all aspects at different levels of abstraction without any ambiguities or potential for misunderstandings. While specification documents in natural languages are prone to misinterpretations, this is not the case with formal specifications which are based on mathematical foundations and unambiguous semantics and syntaxes. The secondary objective is to obtain formally verified (by means of a simulation tool) specifications to provide a solid basis for the implementation phase. The automatically verified designs that result not only increase the inherent dependability of systems by reducing implementation errors they also provide the basis for robust, real-time monitoring of system execution during the operating phase.

Keywords

Formal methods, petri nets, component-based systems, monitoring, probabilistic framework, dependability.

Kurzbeschreibung

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung eines neuen Verfahrens zum nahtlosen Komponentenentwurf und zur Systemüberwachung durch ein einheitliches Modell, das die Anforderungen der Entwicklung von komplexen dynamischen Systemen erfüllt und somit einen Beitrag zum Entwurf verlässlicher Systeme leistet. Hierfür wird die komponentenbasierte Design-Methodologie KobrA eingesetzt, weil diese eine schrittweise Komponentenerlegung auf verschiedenen Abstraktionsebenen und Sichten durchführt. Sie beinhaltet sowohl „Top-down“-Elemente als auch „Bottom-up“-Ansätze, die für eine effiziente prototypische Systemrealisierung geeignet sind.

Mit der Entwicklung eines formalen echtzeitfähigen Überwachungs- und Fehlererkennungsmechanismus wird die KobrA-Methode durch eine formale Modellierungssprache erweitert, welche sowohl für die Softwareentwickler als auch für die Ingenieure verständlich sein soll. Aus diesem Grund sollte diese Sprache eine eindeutige und streng definierte Semantik besitzen. Die einheitliche Beschreibung der Systemkomponenten sowie der Überwachungskomponenten durch denselben formalen Sprachmittel ermöglicht die systematische Einbettung der Überwachung über den gesamten Entwicklungsprozess und dessen Ausführung während des Betriebs.

Petri-Netze gehören zur Graphentheorie und zählen seit mehreren Jahren zu den mächtigsten Spezifikationswerkzeugen in verschiedenen Gebieten. Sie erlauben die Beschreibung des Komponentenverhaltens durch ein Netzwerk, bestehend aus Knoten und aus Bedingungen für den Datenfluss zwischen diesen Knoten. Wesentliche Vorteile von Petri-Netzen sind zum einen ihre formale mathematische Formulierung, die auf einem soliden theoretischen Fundament beruht, sowie zum anderen die explizite Abbildung des Prozesszustandes über ein Markierungskonzept. Petri-Netze ermöglichen zusätzlich die Darstellung sequentieller, sich gegenseitig ausschließender sowie paralleler Aktivitäten, die Modellierung und Visualisierung von Systemverhalten sowie die Nebenläufigkeit und die Synchronisation von kooperativen Prozessen.

In dieser Arbeit erfolgt die Verhaltensbeschreibung der Überwachungskomponenten durch eine neue Klasse von Petri-Netzen, so genannte „Modifizierte Partikel Petri-Netze“ (engl., Modified Particle Petri Nets „MPPN“). Diese Netzklasse beinhaltet hybride Petri-Netze für die Modellierung des hybriden Systemverhaltens und einen Partikelfilter als probabilistische Erweiterung, um die Überwachung als Tracking-Problem aufzufassen. Petri-Netze bieten eine

vollständige und konsistente Beschreibung der Prozesse, die graphische Anschauung sowie Simulation und Animation als Testmöglichkeit bereits während der Entwurfsphase. Die Kombination aus Kobra-Beschreibungsformalismus und Petri-Netzen erlaubt eine anschauliche, modular und hierarchisch strukturierte Modellierung, direkt in einer formalen Sprache.

Durch unterstützende Werkzeuge, die im Rahmen dieser Arbeit entwickelt sind, kann die Realisierung der Überwachungskomponente direkt aus der Spezifikation generiert werden. Hierfür wird das Petri-Netzmodell in ein textuelles kompaktes XML-Austauschformat (engl., „Extensible Markup Language“) transformiert, welche sich an dem PNML-Standard (engl., „Petri Net Markup Language“) orientiert. Diese generische Vorlage enthält das Komponentenverhalten und die für den Überwachungsprozess notwendigen Parameter.

Der besondere Aspekt für den Einsatz derselben formalen Methode, nämlich die Petri-Netze, sowohl für die Spezifikation als auch für die Realisierung, beruht auf zwei Zielen. Das primäre Ziel ist, ein einheitliches verständliches Ausdrucksmittel für die Entwurfsphase eines Systems zu stellen, mit dem alle Aspekte des ausgewählten Abstraktionsniveaus unmissverständlich dargestellt werden können. Denn Spezifikationsdokumente in natürlichen Sprachen sind anfällig für Missverständnisse, während formale Spezifikationen auf mathematischen Beschreibungen und eindeutiger Semantik und Syntaxen basieren. Das sekundäre Ziel ist eine formale überprüfbare Spezifikation (mittels eines Simulationswerkzeuges) als solide Basis für die Realisierungsphase zu bilden. Denn eine automatisch verifikationsbasierte Systementwicklung stellt eine Möglichkeit zur Erhöhung der Systemverlässlichkeit dar. Die andere Möglichkeit basiert auf der Robustheit des Überwachungsverfahrens während der Betriebsphase.

Schlagworte

Formale Methoden, Petri-Netze, Komponentenbasierte Systeme, Überwachung, Probabilistisches Framework, Verlässlichkeit.

INHALTSVERZEICHNIS

1	Einleitung	21
1.1	Motivation und Problemformulierung.....	21
1.2	Einordnung des Lösungsansatzes.....	22
1.3	Kernbeiträge der Arbeit.....	26
1.4	Gliederung der Arbeit.....	30
1.5	Eigene Publikationen.....	33
2	Theoretische Grundlagen der Systementwicklung und Überwachung	35
2.1	Komponentenbasierte Systementwicklung	35
2.1.1	Stand der Technik von Komponentenmodellen?	36
2.1.2	Der Ansatz Kobra.....	38
2.1.3	Formale Beschreibung der Komponentenverhalten	44
2.1.4	Graphentheorie in Software-/Systemengineering	45
2.2	Grundlagen dynamischer Systemmodellierung.....	47
2.2.1	Dynamische Systembeschreibung.....	48
2.2.2	Systemdynamik und Zustandsübergänge	49
2.2.3	Hybride Modelle.....	52
2.3	Überwachung und Fehlerdiagnose	54
2.3.1	Aufgabe der Überwachung.....	54
2.3.2	Verfahren der Fehlerdiagnose, Defizite und Anforderungen	56
2.4	Defizite im Stand der Technik.....	59
2.4.1	Multi-Model Filter.....	59

2.4.2	Hybride bayesische Netze	60
2.4.3	Diskrete Modelle und numerische Filter	60
2.5	Probabilistische Zustandsschätzung	61
2.5.1	Sequentielles Monte-Carlo-Sampling	64
2.5.2	Partikel-Filter.....	67
2.6	Petri-Netze für die Modellierung	69
2.6.1	Grundlegende Elemente von Petri-Netzen	69
2.6.2	Modellierung dynamischer Verhalten durch Petri-Netze.....	72
2.6.3	Diskrete, kontinuierliche und hybride Petri-Netze.....	74
2.6.4	Possibilistische Petri-Netze	77
2.6.5	Vergleich von Petri-Netzen mit endlichen Automaten	80
3	Einheitlicher heitlicher Ansatz zur hybriden Modellierung und Überwachung	83
3.1	Modifizierte Partikel-Petri-Netze (MPPN)	83
3.1.1	Definition der MPPN.....	84
3.1.2	Dynamik der MPPN	86
3.1.3	Gesamtstruktur der MPPN-Prozesse	88
3.1.4	Prädiktionsschritt im MPPN.....	90
3.1.5	Korrekturschritt im MPPN	94
3.2	Fehlerdiagnose.....	99
3.3	Architektur des Ansatzes.....	102
3.4	Hierarchische Überwachung mittels MPPN.....	104
4	Einbettung der Überwachung in die Systementwicklung	109
4.1	Das Konzept der systematischen Entwicklung einer Überwachung ..	109
4.2	Verhaltensüberwachung innerhalb verschachtelten Strukturen	112

4.3	Formaler Überwachungsentwurf in KobrA.....	120
4.4	Überwachungsrealisierung durch Wiederverwendung von Komponentenbeschreibungen.....	125
4.5	Überwachung in Teststrukturen und Verlässlichkeitsanalyse.....	126
4.6	Beispiele	127
5	Entwicklungs-, Simulations- und Analyseumgebung	133
5.1	Komponenten des Überwachungswerkzeugs.....	133
5.2	Auswahl und Erweiterung des Petri-Netzeditors	134
5.3	Der Netzcompiler und Export in XML-Format	144
5.4	Der MPPN Simulator	145
5.4.1	Variablen, Funktionen und Prozesse	146
5.4.2	Prädiktion der Markierung (Marking Prediction)	148
5.4.3	Prädiktion der Partikel (Particle Prediction)	153
5.4.4	Korrekturschritt (Correction step).....	153
6	Fallstudie: Überwachung der Navigation autonomer mobiler Roboter in einer Indoor-Umgebung.....	157
6.1	Arbeitsrelevante Grundlagen der Navigation mobiler Roboter	157
6.1.1	Umgebungsmodellierung	157
6.1.2	Relative und absolute Selbstlokalisierung.....	160
6.1.3	Pfadplanung.....	161
6.1.4	Die Anwendung: Autonomer Rollstuhl.....	161
6.2	Entwicklung der Überwachungsstruktur	163
6.2.1	Das Überwachungskonzept.....	163
6.2.2	Eingesetzte Sensorsysteme für die Überwachung.....	165
6.2.3	Ein kombiniertes Umgebungsmodell als GUI.....	169

6.2.4	Definition von Weg-, Missions- und Bypasspunkten in der Steuerungsstruktur	172
6.3	Implementierung und Integration des Überwachungssystems	174
6.4	Navigationsüberwachung mittels MPPN	176
7	Evaluierung des Verfahrens.....	187
7.1	Robustheit und Empfindlichkeit.....	187
7.2	Reduktion der Komplexität durch Unterräumen	193
7.3	Usability	196
8	Fazit und Ausblick	202
	Liste der Abkürzungen.....	207
	Literaturverzeichnis	208

Abbildungsverzeichnis

Abbildung 1.3.1: Konzept der On-line Überwachung und Fehlererkennung	28
Abbildung 1.3.2: Struktur der Entwicklungsumgebung für die Überwachung.....	30
Abbildung 2.1.1: Komponentenentwicklung und -zusammensetzung in Kobra	36
Abbildung 2.1.2: Drei Sichtweisen einer Komponente.....	39
Abbildung 2.1.3: Modellbasierte Komponentenbeschreibung nach Kobra [Atk03a].....	40
Abbildung 2.1.4: Konzept der Komponentenerlegung in Kobra	40
Abbildung 2.1.5: Zerlegung eines Fahrzeugsteuerungssystems in Komponenten.....	41
Abbildung 2.1.6: Strukturelle Spezifikation der Komponente „Geschwindigkeitsregelung“	41
Abbildung 2.1.7: Strukturelle Realisierung der Komponente „Geschwindigkeitsregelung“	41
Abbildung 2.1.8: Strukturelle Spezifikation der Komponente „Room“	42
Abbildung 2.1.9: Funktionale Spezifikation der Komponente „Thermostat“	42
Abbildung 2.1.10 Statechart und Aktivitätsdiagramm für die Verhaltensspezifikation und -realisierung	43
Abbildung 2.1.11: Ungerichteter Graph.....	46
Abbildung 2.1.12: Gerichteter Graph.....	46
Abbildung 2.2.1: Klassen der dynamischen Systeme	47
Abbildung 2.2.2: Signalraum eines dynamischen Systems.....	48
Abbildung 2.2.3: Zustandsraum eines dynamischen Systems	49
Abbildung 2.2.4: Hybride Systeme als Kombination der kontinuierlichen und diskreten Dynamik.....	51
Abbildung 2.2.5: Hybride Automaten.....	52
Abbildung 2.2.6: Hybride Statecharts	53
Abbildung 2.2.7: Hybride Petri-Netze	54
Abbildung 2.3.1: Überwachung und Fehlerdiagnose.....	56
Abbildung 2.3.2: Quantitative Prozessüberwachung	58
Abbildung 2.5.1: Das Tracking-Problem	63
Abbildung 2.5.2: Prozesse der Zustandsschätzung.....	64
Abbildung 2.5.3: Sequential Importance Sampling Prozess [Fun04]	66
Abbildung 2.5.4: Algorithmus für Sequential Importance Sampling (SIS).....	66
Abbildung 2.5.5: Streuung der Partikel über die Zeit	67
Abbildung 2.5.6: Algorithmus für Sampling Importance Resampling (SIR)	68

Abbildung 2.5.7: Das Prinzip des SIR	68
Abbildung 2.6.1: Grundlegende Elemente eines Petri-Netzes	70
Abbildung 2.6.2: Unmarkiertes Petri-Netz	71
Abbildung 2.6.3: Markierung, Vorbereich und Nachbereich eines Petri-Netzes	73
Abbildung 2.6.4: Markierung nach dem Schalten einer Transition	73
Abbildung 2.6.5: Erreichbarkeitsgraph eines Petri-Netzes [Dav05]	74
Abbildung 2.6.6: Diskrete (links) und kontinuierliche (rechts) Petri-Netze [Dav05]	75
Abbildung 2.6.7: Hybride Petri-Netze [Dav05]	75
Abbildung 2.6.8: Einfluss der diskreten Komponente auf die kontinuierliche Komponente	76
Abbildung 2.6.9: Einfluss der kontinuierlichen Komponente auf die kontinuierliche Komponente	77
Abbildung 2.6.10: Konvertierung von kontinuierlicher in diskrete Markierung und umgekehrt	77
Abbildung 2.6.11: Überwachung der Prozessdurchführung an Flugzeugen	80
Abbildung 2.6.12: Endlicher Automat	82
Abbildung 3.1.1: Hybride Markierung eines MPPN	85
Abbildung 3.1.2: Das Feuern einer symbolischen Transition	87
Abbildung 3.1.3: Das Feuern einer numerischen Transition	87
Abbildung 3.1.4: Das Feuern einer hybriden Transition	88
Abbildung 3.1.5: Phasen der Schätzung	89
Abbildung 3.1.6: Schätzungsprozess der MPPN	90
Abbildung 3.1.7: MPPN-Modell des Thermostaten	91
Abbildung 3.1.8: Berechnung der Endmarkierung	92
Abbildung 3.1.9: Prädiktion der Partikel	94
Abbildung 3.1.10: Korrektur der Prädiktion	95
Abbildung 3.1.11: Gruppierung der Partikel und Klassenbildung	96
Abbildung 3.1.12: Wahrscheinlichkeiten an numerischen und symbolischen Stellen	96
Abbildung 3.1.13: Ergebnisse eines Simulationsschrittes	97
Abbildung 3.1.14: Resampling	97
Abbildung 3.2.1: Residuumgenerierung	100
Abbildung 3.2.2: Schematische Darstellung des entwickelten Fehlerdiagnosemechanismus	101
Abbildung 3.2.3: Modellierung der Unsicherheit durch Prozessrauschen (a) (aus der Spezifikation sp) und Messrauschen (b) (aus der Messung m)	102

Abbildung 3.3.1: Schematische Darstellung der Wechselwirkungen im MPPN.....	103
Abbildung 3.4.1: Hierarchische Überwachung mittels MPPN	105
Abbildung 3.4.2: Zustandsraummodell einer hierarchischen Überwachung	106
Abbildung 3.4.3: Konzept zur hierarchischen Überwachung und Rekonfiguration	107
Abbildung 4.1.1: Integration der Überwachung in der Entwicklung und Evaluation dynamischer Systeme	110
Abbildung 4.1.2: MPPN und Kobra in der Spezifikations- und Realisierungsphase	111
Abbildung 4.2.1: Funktionale und verhaltensbasierte Zerlegung	112
Abbildung 4.2.2: Die Verhaltensebenen als gekoppelte dynamische Systeme.....	113
Abbildung 4.2.3: Das Konzept der Verschachtelung	115
Abbildung 4.2.4: Temperaturregelung in einem Haus	115
Abbildung 4.2.5: Die Temperaturentwicklung in den Räumen	116
Abbildung 4.2.6: Hierarchische Zerlegung des HTCS	117
Abbildung 4.2.7: Demonstrations-Plattform zum HCS	118
Abbildung 4.2.8: HCS-Komponentenübersicht	118
Abbildung 4.2.9: Die verschachtelten Überwachungsebenen im HCS.....	119
Abbildung 4.2.10: Beschreibung der Überwachung im HCS mittels Klassendiagrammen.....	119
Abbildung 4.3.1: Formale Beschreibung des Thermostat-Überwachers	120
Abbildung 4.3.2: Formale Beschreibung des Heizkörper-Überwachers.....	120
Abbildung 4.3.3: Spezifikation der Überwachungskomponente	121
Abbildung 4.3.4: Realisierung der Überwachungskomponente.....	121
Abbildung 4.3.5: Struktur des hybriden Überwachers und Schnittstellen zum System	123
Abbildung 4.3.6: Textuelle Vorlage zur generischen Beschreibung der Überwacher-Struktur	124
Abbildung 4.4.1: XML-Vorlage im Überwachungsprozess	125
Abbildung 4.4.2: Generierung des XML-BDF aus dem MPPN-Modell des Überwachers	126
Abbildung 4.5.1: Datenaustausch mit dem Test-Sheet für Verlässlichkeit.....	126
Abbildung 4.5.2: Konzept der Integration in Teststrukturen	127
Abbildung 4.6.1: Die Komponenten des HTCS.....	128
Abbildung 4.6.2: Übersicht der Schnittstellen und Komponentenklassen.....	128
Abbildung 4.6.3: Übersicht der Schnittstellen und Komponentenklassen.....	129
Abbildung 4.6.4: Schätzung der Betriebsmodi	129

Abbildung 4.6.5: TestSheet für die Temperaturentwicklung im Raum	130
Abbildung 4.6.6: Navigation eines Rollstuhls	130
Abbildung 4.6.7: Der Überwacher für die Orientierungsänderung als MPPN und BDF.....	131
Abbildung 4.6.8: Schätzungsergebnisse im fehlerfreien Fall	131
Abbildung 4.6.9: Schätzungsergebnisse im Fall einer Abweichung von der Spezifikation	132
Abbildung 4.6.10: Versorgung des TestSheets durch Überwachungsdaten.	132
Abbildung 5.1.1: Entwicklungsumgebung der Überwachung	133
Abbildung 5.2.1: EXHOST-PIPE Benutzeroberfläche	135
Abbildung 5.2.2: Struktur der Entwicklungsumgebung.....	136
Abbildung 5.2.3: Graphische Bedienoberfläche von EXT-PIPE.....	137
Abbildung 5.2.4: Erweiterungen an numerischen Transitionen.....	137
Abbildung 5.2.5: Test-And Bedingung	138
Abbildung 5.2.6: Test-Compare Bedingung	138
Abbildung 5.2.7: Test-Interval Bedingung	138
Abbildung 5.2.8: Test-Vector Bedingung	138
Abbildung 5.2.9: Fusion-Sets für hierarchische Petri-Netze.....	139
Abbildung 5.2.10: Austausch von Token zwischen hierarchischen Petri-Netzen	139
Abbildung 5.2.11: Möglichkeiten der Implementierung von Fusion-Sets.....	140
Abbildung 5.2.12: Differenzgleichung mit zeitvariantem Eingang	140
Abbildung 5.2.13: Implementierung der Schnittstellen für socketsbasierte Simulation.....	141
Abbildung 5.2.14: Socketverbindung und -konfiguration	141
Abbildung 5.2.15: Echtzeitsimulation mit dem realen System.....	142
Abbildung 5.2.16: Ergebnisse bei einer off-line Simulation.....	143
Abbildung 5.2.17: Ergebnisse bei einer on-line Simulation	143
Abbildung 5.3.1: Schritte der Überwachungsentwicklung und -simulation	144
Abbildung 5.3.2: Sequenzdiagramm zur Erstellung einer Netzdatei	145
Abbildung 6.1.1: Beispiel einer geometrischen Karte	158
Abbildung 6.1.2: Beispiel einer topologischen Karte	159
Abbildung 6.1.3: Generierung einer hybriden Karte.....	159
Abbildung 6.1.4: Die Pfadplanung mittels Zwischenzielen.....	161
Abbildung 6.1.5: Ein autonomer elektrischer Rollstuhl.....	162

Abbildung 6.1.6: Ein autonomer elektrischer Rollstuhl.....	162
Abbildung 6.2.1: Die Verhaltensebenen Pfadplanung und Positionsregelung	163
Abbildung 6.2.2: Die Verhaltensebenen der Geschwindigkeitsregelung	164
Abbildung 6.2.3: Ermittlung der zurückgelegten Strecke [Bad91].....	165
Abbildung 6.2.4: Aufbau eines RFID-Transponders	167
Abbildung 6.2.5: Anbindung des RFID-Systems am Rollstuhl.....	167
Abbildung 6.2.6: Das Schreiben der Position der ersten Knoten.....	168
Abbildung 6.2.7: Header Format.....	168
Abbildung 6.2.8: AVP Format	168
Abbildung 6.2.9: Datenaustausch im RFID-System ([Mab12])	169
Abbildung 6.2.10: Metrische Darstellung der Testumgebung [Ale11].....	169
Abbildung 6.2.11: Topologische Darstellung der Testumgebung [Ale11].....	170
Abbildung 6.2.12: Platzierung der RFID-Tags in der Umgebung	171
Abbildung 6.2.13: Ermittlung des Aktionsraumes.....	171
Abbildung 6.2.14: Das GUI für die Navigation	172
Abbildung 6.2.15: Anfahren und Annähern von Objekten mit Hilfe der Bypasspunkte	173
Abbildung 6.2.16: „Waypoints“ und „Bypasspoints“ als Konfigurationsdatei	174
Abbildung 6.3.1: Anbindung des Überwachungssystems an den Rollstuhl	175
Abbildung 6.4.1: Überwachungsmodell der Positionsregelungsebene.....	176
Abbildung 6.4.2: Prozess der Positionsschätzung.....	178
Abbildung 6.4.3: Algorithmus zur probabilistischen Lokalisierung im numerischen Modell.....	179
Abbildung 6.4.4: Trajektorie zum fehlerfreien Navigationszenario	179
Abbildung 6.4.5: Das hybride Überwachungsmodell der Pfadplanung.....	180
Abbildung 6.4.6: Das BDF zur Trajektorie 1-3-4-7-8-9	182
Abbildung 6.4.7: Ergebnisse der numerischen Schätzung der „x-Position“.....	182
Abbildung 6.4.8: Ergebnisse der numerischen Schätzung der „y-Position“.....	183
Abbildung 6.4.9: Ergebnisse der symbolischen Schätzung	184
Abbildung 6.4.10: Ergebnisse der Konsistenzanalyse	185
Abbildung 6.4.11: Überwachungsergebnisse bei Abweichungen innerhalb des spezifizierten Bereichs....	186
Abbildung 6.4.12: Ergebnisse der numerischen Schätzung für die x-Position und die y-Position.....	186
Abbildung 6.4.13: Ergebnisse der symbolischen Schätzung für die x-Position und die y-Position.....	186

Abbildung 7.1.1: Experiment zum inkonsistenten Navigationsverhalten.....	187
Abbildung 7.1.2: Detektion von Inkonsistenzen.....	188
Abbildung 7.1.3: Ergebnisse der Konsistenzanalyse.....	188
Abbildung 7.1.4: Experiment zur Überschreitung der Sicherheitsgrenze.....	189
Abbildung 7.1.5: Experiment zur kurzzeitigen Störung.....	190
Abbildung 7.1.6: Experiment zum Komponentenfehler.....	191
Abbildung 7.1.7: Empfindlichkeit beim Zustandswechsel.....	192
Abbildung 7.1.8: Empfindlichkeit bei Abweichungen der kontinuierlichen Größen.....	192
Abbildung 7.2.1: Gitterbasierte Navigation ([Feh04]).....	193
Abbildung 7.2.2: Modellierung der Gitterbasierten Navigation mittels Petrinetzen in ([Les06]).....	194
Abbildung 7.2.3: Inzidenzmatrix einer Navigationsinstanz der Größe 3x3.....	194
Abbildung 7.2.4: MPPN basierte Modellierung einer Navigationsinstanz der Größe 3x3.....	195
Abbildung 7.2.5: Beispiele von Navigationsinstanzen zur Berechnung der MPPN-Komplexität.....	196
Abbildung 7.3.1: Planung einer Flugmission.....	197
Abbildung 7.3.2: Missionspunkte für die Überwachung.....	198
Abbildung 7.3.3: Modellierung der Detektionsbereiche.....	198
Abbildung 7.3.4: MPPN-Modell für die Flugüberwachung.....	199
Abbildung 7.3.5: Integration des Assoziativspeichers im Überwachungskonzept ([Ale12]).....	200
Abbildung 7.3.6: Realisierung der Assoziativspeicher ([Ale12]).....	200
Abbildung 7.3.7 Rolle des Assoziativspeichers im Fehlerdiagnosemodul.....	201

Tabellenverzeichnis

Tabelle 2.1.1: Diagramme und Techniken, je nach Sicht und Phase	44
Tabelle 2.1.2 TestSheet in Kobra.....	44
Tabelle 2.6.1: Aufbau von Petri-Netzen.....	71
Tabelle 3.1.1: Graphische Elemente der MPPN	85

1 Einleitung

1.1 Motivation und Problemformulierung

Echtzeitanforderungen, die Vernetzung von Geräten, die große Anzahl von Systemkomponenten und sicherheitskritische Aspekte erhöhen heutzutage die Komplexität technischer Systeme. Die starke Interaktion dieser Systeme miteinander und mit der Umgebung macht sie auch für Fehler sehr anfällig. Trotz dieser zunehmenden Komplexität und Fehleranfälligkeit, sind durch unsere gesteigerte Abhängigkeit von den technischen Systemen die Erwartungen an deren Zuverlässigkeit, Sicherheit und Performanz gestiegen. Somit wünschen wir uns heutzutage ein System, das trotz auftretender Fehler in einigen Komponenten funktionsfähig bleibt und die spezifizierten Dienste weiterhin anbieten kann. Wichtig dabei ist, dass das System immer in einem „sicheren“ Zustand verweilt. Derartige Systeme werden als „fehlertolerant“ oder „zuverlässig“ bezeichnet ([Bel86]). Durch eine frühzeitige Fehlererkennung können geeignete weiterführende Maßnahmen getroffen werden, und somit wird die Verursachung von Schäden im System oder in seiner Umwelt verhindert. Um dies zu erreichen sollen Sicherheitsaspekte und Überwachungsmechanismen sowohl in einer frühen Phase der Systementwicklung als auch während des Betriebs eingebettet werden.

Diese Komplexität und Fehleranfälligkeit in Software- und Systemengineering erfordert eine Vereinigung von Paradigmen, welche gleichzeitig ein großes Problem abstrahieren und in kleinere Teile zerlegen können sowie Sicherheitsmechanismen schon in der Entwurfsphase einbetten können. Eine weitere Herausforderung heutiger komplexer dynamischer Systeme ist deren hybrides Verhalten. Diese Systeme bestehen meistens aus Komponenten mit kontinuierlichem Verhalten und Komponenten mit diskretem Verhalten, die in Wechselwirkungen zueinander stehen, und sie bilden somit die Klasse der so genannten „Hybriden Systeme“ ([Ant00]). Das wesentliche Merkmal dieser Vereinigung ist die Wechselwirkung beider Teile der Systemdynamik, für die bisher in der Literatur keine einheitliche Beschreibung vorzufinden ist. Es ist deswegen notwendig, Methoden zu entwickeln, die das Verhalten solcher Systeme in der richtigen Weise behandeln, d. h. Methoden, die sowohl das kontinuierliche Verhalten als auch diskrete Verhalten und ihre Interaktionen betrachten.

Die Problemstellung in dieser Arbeit lässt sich in drei Teilprobleme zerlegen, welche gleichzeitig behandelt werden müssen, und zwar nach einer Systematik, die als Guideline für die Entwicklung verlässlicher hybrider komplexer Systeme vorgeschlagen wird.

Problem I: Einbettung der Überwachung im Entwicklungsprozess und die einheitliche Beschreibung sowie Realisierung dieser Sicherheitskomponente sowie anderen Systemkomponenten. Somit stellt sich die Frage nach einer geeigneten Entwicklungsmethode, die die nahtlose Integration von Sicherheitsmechanismen unterstützt.

Problem II: Konzeption und Entwicklung eines geeigneten Verfahrens zur Überwachung des Verhaltens hybrider komplexer dynamischer Systeme und zur frühzeitigen Fehlererkennung. Dies erfordert die Beschäftigung mit den Fragestellungen der hybriden Modellierung bei der Verhaltensbeschreibung, der Topologie für die Modularisierung und Hierarchisierung und des Abstraktionsgrades der Modelle.

Problem III: Aufgrund des Einsatzes von formalen Methoden, sowohl in der Entwicklungsphase für Verifikations- und Validierungszwecke als auch während des Betriebs für die On-line Analyse, werden Software-Werkzeuge benötigt. Diese werden im Rahmen dieser Arbeit, unter Berücksichtigung der Anforderungen an sicherheitskritische Systeme, wie z. B. die Echtzeitfähigkeit, entwickelt.

1.2 Einordnung des Lösungsansatzes

Wichtig für Entwicklung einer gemeinsamen Lösung für die oben genannten Probleme ist die Auswahl einer geeigneten Entwurfsmethode, die eine nahtlose Integration weiterer Konzepte erlaubt sowie die Fähigkeit besitzt, verschiedene Abstraktionsebenen der dynamischen Modellierung und Systemzerlegung zu unterstützen. Die nahtlose komponentenbasierte Entwurfsmethode Kobra „Komponentenbasierte Anwendungsentwicklung“ ([Atk00]-[Atk03a]-[Atk05]-[Atk07]) meistert diese Herausforderung durch die gemeinsame Nutzung der modellgetriebenen und komponentenbasierten Entwicklungsparadigmen, die bisher nur getrennt entwickelt und angewendet wurden. Ein weiterer Vorteil der Kobra-Methode ist, dass sie sich auf Sicherheitskonzepte erweitern lässt, z. B. Durch die Definition des Built-in-Tests schon im Entwurfsprozess ([Atk02b]-[Atk03b]-[Bre07]-[Atk08]-[Atk10]). Dies ermöglicht die Integration von Sicherheitsmechanismen, wie z. B. die Überwachung von Teststrukturen in den frühen Phasen der Entwicklung. Zur Behandlung der Komplexität verwendet Kobra einen kombinierten Ansatz aus Top-Down Entwurf und Bottom-up Implementierung ([Atk14]). Diese ist kompatibel mit erfolgreich eingesetzten Systemarchitekturen, die bei sicherheitskritischen Anwendungen verwendet werden ([Ble05]-[Bar09]-[Kan10]-[Wag10]-[Mek13]-[Ale14]). Durch die Kompatibilität von Kobra mit

diesen verlässlichen Architekturen entstehen statt eines komplexen Systems mehrere verlässliche Teilsysteme von geringerer Komplexität.

⇒ Diese Stärken machen Kobra attraktiv für diese Arbeit.

In der Literatur ist der Begriff „Fehler“ sehr unterschiedlich interpretierbar. Allerdings zeigen alle Arten von Fehlern eine Gemeinsamkeit auf: die Abweichung eines Ist-Wertes von einem Soll-Wert ([Abd06]). Damit ist der Begriff „Fehler“ als eine Abweichung vom gewünschten Verhalten einer Komponente definiert, was oftmals mittels einer Spezifikation postuliert wird. Dieses Verhaltensmodell des Systems stellt den Kern der modellbasierten Überwachung dar. Typische einheitliche Modellierungsdarstellungen, die in der Literatur behandelt werden, basieren auf Hybriden Automaten (HA) und Hybriden Petri-Netzen (HPN). Letzteres scheint besser geeignet, aufgrund mehrerer Vorteile, wie Kompaktheit, Modularität und der Fähigkeit, die Parallelität der Prozesse zu modellieren. Petri-Netze werden heute zur realitätsnahen Modellierung und Analyse von komplexen Systemen in verschiedenen Bereichen zunehmend angewendet ([Den04]-[Bal11]-[Dur12]-[Sun15]). Der Einsatz formaler Methoden zur Analyse und Verifikation, besonders bei der Entwicklung sicherheitskritischer Applikationen, wird in verschiedenen Forschungsarbeiten intensiv untersucht ([Bar92]-[Sun15]). Daher soll man eine Methode einsetzen, welche Eigenschaften, wie z. B. Analysierbarkeit, Eindeutigkeit, Vollständigkeit, Konsistenz und Korrektheit, besitzt. Besonders wichtig ist, dass sie Untersuchungs- und Verifikationsmöglichkeiten bietet, insbesondere:

- Die Ermittlung kausaler Systemeigenschaften, die zur funktionalen Korrektheit des Systems beitragen, wie z. B. die Erreichbarkeit gewünschter Zustände, Lebendigkeit, Deadlocksfreiheit, Nebenläufigkeit.
- Die Ermittlung zeitlich veränderlicher technischer Qualitätsaspekte des Systems, wie Systemleistung (Performance) und Verlässlichkeit (Dependability).
- Die Überprüfung der Korrektheit der Spezifikation, z. B. durch Erkennung von Konflikten.
- Die Lieferung einer eindeutigen Interpretation, die auch für Personen ohne Kenntnisse im Bereich der Softwareentwicklung verständlich sein kann.
- Die Ausführbarkeit: Eine Systembeschreibung soll automatisch von einer Spezifikation zu einer (Teil-)Implementierung umgesetzt werden.

Solche Testverfahren und automatische Analyse der Konsistenz sowie der Überwachung von Komponentenverhalten sollen zum Bestandteil heutiger Systeme werden. Petri-Netze als Spezifikationsmethode erfüllen diese Anforderungen besonders gut, und durch die in den letzten Jahren entstandenen Erweiterungen ist die Mächtigkeit von Petri-Netzen in Bezug auf die Analysierbarkeit gestiegen. In Bezug auf die Systementwicklung liegt die Mächtigkeit von Petri-Netzen darin, dass sie für die Beschreibung nebenläufiger Prozesse sowie für komplexe hierarchische Systeme besonders geeignet sind. Somit können die verschiedenen Subsysteme auf unterschiedlichen Detaillierungsgraden mit demselben Beschreibungsmittel spezifiziert werden.

⇒ *Einsatz von Petri-Netzen als formale Methode in der Entwicklung sicherheitskritischer Systeme.*

Der Einsatz von Werkzeugen zur automatischen Verifikation ist ein weiterer Vorteil formaler Methoden. Petri-Netze als werkzeuggestützte Techniken bleiben sicherlich auch zukünftig interessant, weil die Wissenschaft ständig an der Entwicklung von neuen Arten von Petri-Netzen sowie an fortgeschrittenen Petri-Netz-Tools arbeitet ([PNT15]).

⇒ *Für die automatische Verifikation und Validierung ist die Entwicklung eines Petri-Netzeditors bzw. Petri-Netz Simulators erforderlich.*

Den dynamischen hybriden Systemen ist gemein, dass die vorliegenden Sensormessungen verrauscht sind und nur mit einer gewissen Unschärfe Aussagen über den Systemzustand gemacht werden können. Ungenaue Modelle bilden eine weitere Ursache für den ungenauen und begrenzten Rückschluss auf den Zustand. Ein optimales Rahmenkonzept zur Betrachtung der Unsicherheit bietet die probabilistische Analyse durch den Einsatz von probabilistischen Filtern, da sie rekursiv funktionieren und damit eine genauere Schätzung des Systemzustands liefern.

⇒ *Der probabilistische Framework ist eine Hauptanforderung des zu entwickelnden Überwachungsverfahrens.*

In den letzten drei Jahrzehnten wurde eine Menge Forschung im Gebiet der Überwachung und Fehlerdiagnose hybrider dynamischer Systeme durchgeführt ([Bou97]-[Bal02]-[Bir06]-[Pet06]-[Bar12]). Einige basieren auf numerischen Filtern, wie z. B. Multi-Modellen von Kalman-Filtern oder Partikel-Filtern ([Fun04]). Andere Ansätze beruhen auf der Kombination von hybriden Automaten mit numerischen Filtern, z. B. hybride probabilistische Automaten mit einem Satz von Kalman-Filtern oder hybride Automaten mit Partikel-Filtern.

Verbreitete Ansätze der Systemüberwachung basieren auf einer Zustandsschätzung ([Goo07]-[Ham09]-[Bar12]), die für hybride Systeme eine interessante Herausforderung stellt, insbesondere wenn eine Schätzung sowohl der diskreten als auch der kontinuierlichen Zustände benötigt wird. Ein Blick in die Literatur zeigt, dass mehrere hybride Zustandsschätzverfahren entwickelt wurden ([Hof04]). Der Bereich Robotik verwendet regressionsbasierte Verfahren, neuronale Netze oder so genannte Bayessche Filter ([Hau05]), wie Partikel-Filter und erweiterte Kalman-Filter. Zur Zustandsbestimmung von mobilen Robotern ist der Partikel-Filter schon länger bekannt, z. B. wurden sowohl zur relativen als auch zur absoluten Positionsschätzung SMC Methoden (Sequentielle Monte Carlo Methoden) erfolgreich eingesetzt ([Dea02]-[Ric02]-[Ver04]-[Ioa04]). Dabei ist das Wissen über das beobachtete System zu einem bestimmten Zeitpunkt kein exakter Wert, sondern ein Bereich, der der Schätzung entspricht und mit einer Funktion der Wahrscheinlichkeitsdichte abgebildet wird. Andere Filter, wie z. B. Kalman-Filter wurden nur für die relative Lokalisierung eingesetzt ([Rus04]-[Thr05]). Dabei ist die Roboterposition durch eine gaußsche Wahrscheinlichkeitsverteilung repräsentiert. Die Entscheidung für Partikel-Filter ([Aru02]), vor allem gegenüber dem Kalman-Filter ([Kal60]-[Wel01]), liegt an den Vorteilen, die sie bei vielen realen Problemen zeigen. Insbesondere ermöglicht der Partikel-Filter die Repräsentation multimodaler und nicht-parametrischer Verteilungen, die Analyse nicht-linearer und nicht-Gaußscher dynamischer Systeme. Ein weiterer Vorteil ist die Anpassbarkeit der Schätzgenauigkeit der Prognose. Diese kann jederzeit an die verfügbare Rechnerkapazität angepasst werden ([Ver04]). Partikel-Filter betrachten nicht den gesamten Zustandsraum, sondern sie berechnen die Wahrscheinlichkeitsverteilung nur für eine Teilmenge.

⇒ Einsatz von Partikel-Filtern für die Zustandsschätzung

Beide Probleme, die hybride Modellierung und die Berücksichtigung der Unsicherheit, werden in dieser Arbeit durch eine Kombination der Petri-Netze und Partikel-Filter gelöst. Es ergibt sich eine neue Art von Petri-Netzen, so genannte „Modifizierte Partikel-Petri-Netze“ (engl.; „Modified Particle Petri Nets“, MPPN). Eine Untersuchung des Stands der Technik von Ansätzen, die Petri-Netze verwenden und eine hybride Modellierung unter Berücksichtigung der Unsicherheit anstreben, führt zum Ergebnis, dass die MPPN diese Herausforderung am besten meistern. Einige Ansätze, wie z. B. [Gen07] und [Pao14], haben den Diagnoseansatz von [Sam05] mittels Petri-Netzen, anstatt endliche Zustandsautomaten, verwendet und auf ereignisdiskrete Systeme angewendet. Dabei wollten die Autoren von den

Vorteilen der Petri-Netze bezüglich verteilter Modellierung profitieren und haben damit ein verteiltes Diagnosemodul vorgeschlagen. Jedoch hat keiner dieser Ansätze die kontinuierlichen Aspekte sowie die Unsicherheiten im System berücksichtigt. Andere Petri-Netz-basierte Ansätze, die Unsicherheiten berücksichtigt haben ([RuY09]-[Bas09]-[Jia13]), verwenden Petri-Netze mit “Partial Observation” oder stochastische Petri-Netze. Doch bei all diesen Ansätzen wurde der kontinuierliche Aspekt im Modell nicht berücksichtigt. Der einzige Ansatz, der bisher hybride Petri-Netze durch Partikel-Filter erweitert hat sind die Partikel-Petri-Netze von [Les05]. Dieser Ansatz wurde für die Überwachung von Prozessdurchführungen an Flugzeugen eingesetzt, allerdings nur auf der Missionsplanungsebene und nur zum Teil probabilistisch. Die Wechselwirkungen zwischen beiden Teilen der Dynamik wurden nicht berücksichtigt und die Ermittlung des kontinuierlichen und diskreten Zustands wurde separat durchgeführt. Von daher war die Betrachtung dieser klassischen Petri-Netze nur zum Teil nützlich, und deswegen sind fundamentale Erweiterungen bzw. Modifikationen erforderlich. Diese neuen Konzepte zur Modellierung der Wechselwirkungen in der Dynamik, die mathematischen Theorien zur Feuerbarkeit, die reine probabilistische Schätzung sowie die auf entscheidungsfindungsrelationen-basierte Analyse bilden die Grundprinzipien der MPPN. Somit stellen die MPPN für den aktuellen Stand der Technik einen attraktiven Ansatz dar, worauf sich neue Forschungsarbeiten aufbauen ([Gau14a]-[Gau14b]-[Gau15a]-[Gau15b]).

1.3 Kernbeiträge der Arbeit

Diese Arbeit befasst sich mit der Entwicklung eines formalen Ansatzes für die Verhaltensüberwachung von Komponenten in sicherheitskritischen Systemen, der im Entwicklungsprozess integriert wird und die verschiedenen Modellierungs- und Sicherheitsaspekte eines Systems abdeckt. Diese Überwachungs- und Fehlererkennungskomponente (Monitoring and Fault Detection „MFD“-component) wird, ähnlich wie andere Systemkomponenten, in der Entwurfsphase beschrieben und während des Betriebs eingesetzt, um Abweichungen und Inkonsistenzen im Systemverhalten on-line festzustellen. Wesentliche Eigenschaften dieser Sicherheitskomponenten ist die Fähigkeit, das Systemverhalten durch ein anschauliches Modell darzustellen sowie strukturell die Modularität, die Verteilung und die Nebenläufigkeit zu unterstützen.

Die Beiträge dieser Arbeit lassen sich somit in einen theoretischen und einen praktischen Teil sowie in einen Beitrag zur methodischen Entwicklung einordnen:

I. Der Beitrag zur methodischen Entwicklung

Die Kernidee der Überwachung in dieser Arbeit ist der Vergleich des Systemverhaltens während des Betriebs mit seiner Spezifikation, um Abweichungen bzw. Inkonsistenzen festzustellen. Diese Idee wird umgesetzt durch eine systematische Integration der Überwachung als Sicherheitskomponente im Entwicklungsprozess, in die Systemarchitektur, in Teststrukturen und in die Verlässlichkeitsanalyse. In der Entwurfsphase wird das dynamische Verhalten der Komponenten formal durch hybride Petri-Netze spezifiziert. Diese Verhaltensspezifikation wird in der Realisierungsphase in MPPN umgewandelt. Dieser Schritt erfolgt durch die automatische Generierung einer generischen ([W3C15]) XML-basierten Vorlage - die so genannte „Verhaltensbeschreibungsvorlage“ (engl., „Behavior Description File“, BDF), die ohne Konvertierungen direkt durch einen Menschen lesbar ist. Das BDF stellt dem Systementwickler einen Vorschlag für ein einheitliches Dateiformat, sowohl für die verhaltensbasierte Beschreibung der Systemkomponenten als auch für die Überwachungsentwicklung. Diese Realisierungsvorlage enthält sowohl die Beschreibung der zu überwachenden Komponente (engl., „Component under Monitoring“, CUM) als auch Sicherheitselemente aus der Anforderungsspezifikation, die für den Überwachungsprozess notwendig sind.

Die Ideen hinter der Einbettung der formalen Beschreibung der Überwachungskomponente während des gesamten Entwicklungsprozesses sind:

- Das Systemverhalten mit seiner Spezifikation zu vergleichen, um Abweichungen und Inkonsistenzen festzustellen.
- Eine erstellte Spezifikation auf Systemeigenschaften, z. B. auf Verlässlichkeit, überprüfen zu können.
- Weiterhin soll die Wiederverwendung von Komponentenbeschreibungen durch die Überwachungskomponente sowie deren Integration in Teststrukturen ermöglichen, Fehler möglichst früh zu erkennen.

II. Der theoretische Beitrag

Der Hauptbeitrag dieser Arbeit liegt in der Theorie und Implementierung eines Überwachungsverfahrens, das zur Erhöhung der Verlässlichkeit von sicherheitskritischen

Systemen beiträgt. Es handelt sich um einen probabilistischen graphentheoretischen Ansatz, der folgende Lösungen bietet:

Modellierung der Wechselwirkungen zwischen ereignisdiskreten und kontinuierlichen Modellen in einem probabilistischen Framework:

Hierfür wurden die Petri-Netze in einer neuen Form, so genannte „Modifizierte Partikel-Petri-Netze“ MPPN (engl., „Modified Particle Petri Nets“) eingesetzt, um die Interaktion beider Teile der Dynamiken in einem einheitlichen Framework zu beschreiben. Eine formale Beschreibung dieser Interaktion ist durch die Verkopplung der Petri-Netze (als diskrete Modelle) und den Differentialen bzw. Differenzgleichungen (als kontinuierliche Modelle) umgesetzt. Im Weiteren werden diese hybriden Petri-Netze um die Partikel-Filterung erweitert, um den hybriden Systemzustand zu schätzen. Es werden sowohl die numerischen Werte des kontinuierlichen Zustandsvektors als auch die Petri-Netz-Markierung geschätzt. Dieser MPPN-Formalismus ermöglicht die Überwachung des hybriden Systemverhaltens durch eine modellbasierte Schätzung seiner Zustände und unter Berücksichtigung der Unsicherheit an beiden Teilen der Dynamik sowie die Unsicherheit beim Auftreten der Ereignisse.

Gegenüberstellung der Zustandsschätzung und der Erreichbarkeitsanalyse für die Entscheidungsfindung über das Auftreten von Fehlern

Eine der Hauptmerkmale des MPPN-basierten Ansatzes ist die simultane Untersuchung des wertkontinuierlichen und des diskreten Zustands auf Erreichbarkeit. In Bezug auf Fehlererkennung ermöglicht diese hybride Untersuchung die Zusammenstellung eines für Fehler repräsentativen Residuums mit einer Erreichbarkeitsanalyse der Systemzustände, um eine Aussage über das Systemverhalten zu liefern. Gemessene Daten werden mit den erwarteten verglichen (Residuum-Berechnung), und die Zustände werden auf ihre Erreichbarkeit überprüft. Wird bei der hybriden Zustandsschätzung eine Abweichung erkannt, wird diese weiter untersucht (z. B. die Untersuchung auf Schwellwertüberschreitung oder auf unmögliche Zustandswechsel), um Inkonsistenzen festzustellen (Abbildung 1.3.1).

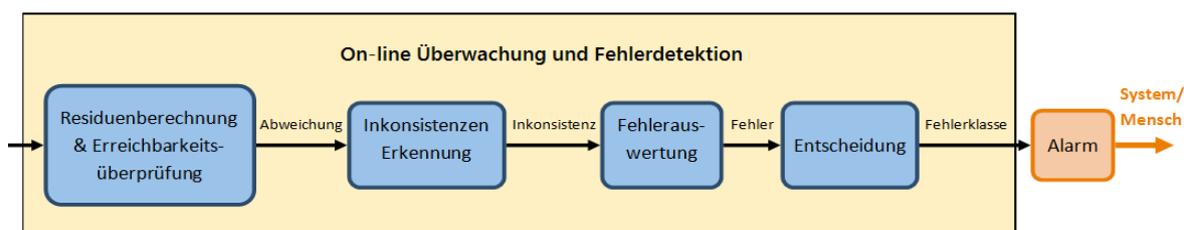


Abbildung 1.3.1: Konzept der On-line Überwachung und Fehlererkennung

Diese Konsistenz wird ausgewertet (z. B. die Auftretensdauer oder Überschreitung kritischer Sicherheitsgrenzen). Wird bestätigt, dass es sich um einen Fehler handelt, wird dieser einer bestimmten Fehlerklasse zugeordnet und dem System oder Nutzer gemeldet

Eine große Bandbreite der Anwendbarkeit:

Für die Erläuterung und Demonstration der Anwendbarkeit des entwickelten Verfahrens, in Zusammenhang mit den einzelnen Konzepten (Spezifikation, Systemarchitektur, Built-in-Test und Verlässlichkeit), werden in den Abschnitten der Arbeit verschiedene technische Systeme als Demonstratoren verwendet. Diese verschiedenen realen Systeme besitzen unterschiedliche Eigenschaften und können somit die Bandbreite des Verfahrens besser aufzeigen. Bei Systemen mit großer Reaktionszeit wird, z. B. anhand eines Temperaturregelungssystems, das so genannte HCS (Heater Control System), vor allem die Theorie der MPPN schnell umgesetzt. Es veranschaulicht die Konzepte der hybriden Modellierung und der Wiederverwendung von Komponentenspezifikationen für die Realisierung der Überwachungskomponente. Für die Applikation des Verfahrens auf komplexen sicherheitskritischen Systemen, bestehend aus Teilsystemen mit unterschiedlichen Reaktionszeiten, wird als Beispielsystem ein autonomer Rollstuhl (aus dem Gebiet „Autonome mobile Roboter“, AMR) verwendet ([Mek11a]- ([Mek11b]). Solch ein System soll zuverlässig und verlässlich arbeiten. Aus diesem Grund wurde in dieser Arbeit die Überwachung in Teststrukturen in Zusammenhang mit der Verlässlichkeitsberechnung integriert, um Aussagen über die Verlässlichkeit ([Lap92]) bzw. Verlässlichkeitsattribute, wie „Safety“ und „Performance“, zu treffen.

III. Der praktische Beitrag

Der praktische Beitrag umfasst zum Teil die Implementierung der im theoretischen Teil entwickelten mathematischen Grundlagen und des aus der Spezifikation und Realisierung abgeleiteten BDFs und zum anderen Teil die Entwicklung eines Analyse- und Simulationswerkzeugs. Hierfür wurden im Rahmen dieser Arbeit ein Petri-Netzeditor (EXT-PIPE), ein Kompilierer (Net-Compiler) und ein MPPN-Simulator (MPPN-Sim) entwickelt (Abbildung 1.3.2). Der Petri-Netzeditor EXT-PIPE bietet den Vorteil der graphischen Animation der entwickelten Petri-Netze und der automatischen Überprüfung einiger Eigenschaften, wie z. B. die Konsistenz, die Richtigkeit der Struktur, die Vollständigkeit, die Lebendigkeit und die Verklemmungsfreiheit der Modelle. Dadurch wird das Auftreten von Fehlern in der Entwurfsphase vermieden. Aus diesen Modellen generiert der Net-Compiler

automatisch den BDF als Spezifikation der Überwachungskomponente. Das generierte BDF wird durch weitere Elemente ergänzt, wie z. B. Unsicherheitsparameter (Rauschen), Randbedingungen (maximale Abweichungen, Grenzwerte usw.) sowie Schätzer-Parameter (Partikelanzahl, Initialisierung) und dem MPPN-Simulator als Realisierung der Überwachungskomponente zur Verfügung gestellt. Dieser führt die Berechnungen für die Systemüberwachung (z. B. hybride Zustandsschätzung, Wahrscheinlichkeitsberechnung, Residuenberechnung, Klassenbildung) on-line durch und übergibt sie dem Fehlerdiagnose-Block. Dieser führt eine Konsistenzanalyse und eine Residuen-Auswertung durch, um eine Aussage über das Auftreten eines Fehlers zu machen.

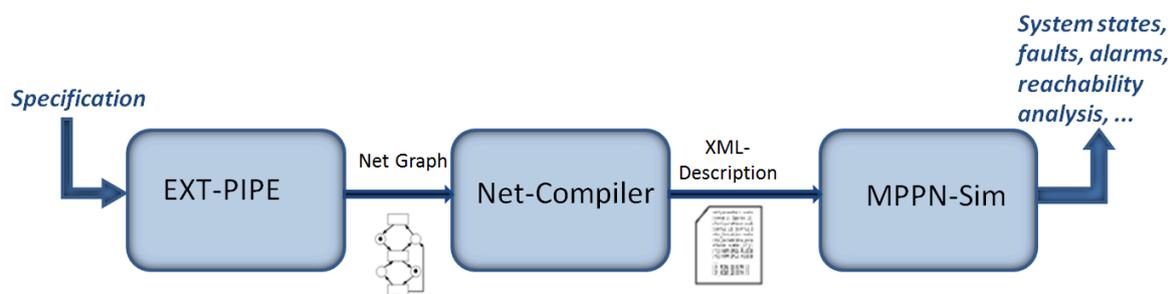


Abbildung 1.3.2: Struktur der Entwicklungsumgebung für die Überwachung

1.4 Gliederung der Arbeit

Im Folgenden wird eine Übersicht zur Gliederung dieser Arbeit und zu den wesentlichen Forschungsbeiträgen gegeben:

- In diesem **einleitenden Kapitel** wurde etwas allgemeiner auf die Motivation der Arbeit eingegangen. Die Beiträge der Arbeit für die Überwachung und Fehlerdiagnose wurden in Zusammenhang mit weiteren Systemkonzepten, wie der Komponentenbeschreibung, Systemzerlegung, Teststrukturen und Verlässlichkeitsevaluation, erläutert.
- **Kapitel 2** beschreibt den Stand der Technik der komponentenbasierten Entwicklung, der Systemüberwachung und der Fehlerdiagnose dynamischer Systeme. Nach einer kurzen Einführung in die hybriden Systeme befasst sich dieses Kapitel im Weiteren mit den erforderlichen theoretischen Grundlagen für die Überwachung dieser Klasse von Systemen sowie der Theorie der Petri-Netze und der Sequentiellen Monte Carlo (SMC)-Filter.

- **Kapitel 3** wird dem in dieser Arbeit entwickelten Überwachungsverfahrensmittel „Modifizierte Partikel-Petri-Netze“, MPPN, gewidmet. Zunächst wird das generische Konzept der Überwachung und dessen Einsatz in hierarchischen Strukturen erläutert. Die MPPN werden mathematisch definiert, und die Regeln für die Schätzungsprozesse werden aufgestellt. Das Konzept der Fehlerdiagnose und der Mechanismus der Konsistenzanalyse, in Abhängigkeit von der wertkontinuierlichen Schätzung und der Erreichbarkeit der diskreten Zustände, wird in Zusammenhang mit den MPPN vorgestellt. Im letzten Teil des Kapitels werden alle oben erwähnten Konzepte und die Durchführbarkeit des Ansatzes am Beispiel des „Heater Control System“, HCS, gezeigt. Dieses HCS ist ein typisches Beispiel für die hybride Modellierung in verschiedenen Literaturen und dient hier der Erklärung von Konzepten. Die Übertragung auf ein reales System der Robotik wird in Kapitel 6 dargestellt, da solche Systeme weitere Anforderungen an die Überwachung stellen.
- In **Kapitel 4** ist die Überwachungskomponente als eine Software-Komponente beschrieben, welche in den verschiedenen Phasen der Systementwicklung berücksichtigt wird. Die systematische Beschreibung, die Integration im Entwicklungsprozess und das Prinzip der Wiederverwendbarkeit der Komponentenbeschreibung für die Überwachung des Komponentenverhaltens werden in diesem Kapitel erläutert. Der Einsatz dieser Komponente, zur Erweiterung der statischen off-line Teststrukturen auf dynamischen on-line Strukturen, um die Systemverlässlichkeit evaluieren zu können, wird anhand von Beispielen gezeigt.
- **Kapitel 5** stellt den praktischen Beitrag in dieser Arbeit dar, nämlich die Werkzeugentwicklung. Es handelt sich hier um eine Entwicklungsumgebung für die Überwachung, deren Komponenten in diesem Kapitel vorgestellt werden. Die Implementierung der MPPN und die mathematischen Grundlagen werden in Pseudocodes beschrieben.
- **Kapitel 6** präsentiert eine Fallstudie, die sich mit der Überwachung der Navigation mobiler Roboter befasst. Das System ist ein autonomer Rollstuhl, der hier als Demonstrationsplattform im Bereich sicherheitskritischer Systeme dient. Die Anwendung der MPPN im Bereich der Robotik benötigt die Verwendung verschiedener Disziplinen der Robotik, der Informatik, der Graphentheorie und der Mathematik. Die dazu verwendeten Grundlagen werden am Anfang des Kapitels kurz

erläutert. Danach wird das entwickelte Überwachungsverfahren auf das Beispiel „Rollstuhl“ angewendet, und zwar ausgehend von der Umgebungsmodellierung und Komponentenbeschreibung, über den Entwurf der verschiedenen Überwachungsmodelle des Navigationsprozesses, bis zur Applikation des Fehlererkennungsmechanismus.

- In **Kapitel 7** wird der entwickelte Ansatz auf verschiedene Kriterien evaluiert.
- **Kapitel 8** fasst die Ziele sowie die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf mögliche zukünftige Erweiterungen.

1.5 Eigene Publikationen

Mekacher Zouaghi L., Koslowski M., Alexopoulos A., Barth F., Jipp M., Fajardo R., Luo Y., Wagner A. und Badreddin E.: "Dependable component-based design on the Example of a Heating Control System." ,Workshop on the Design of Dependable Critical Systems, Hamburg, 2009

Mekacher Zouaghi L., Wagner A. und Badreddin E.: „Hierarchical hybrid monitoring for autonomous systems.“, Workshop on the Design of Dependable Critical Systems, Hamburg 2009

Luo, Y., A. Wagner, L. Mekacher Zouaghi and E. Badreddin: An integrated monitor-diagnosis-reconfiguration scheme for (semi-) autonomous mobile systems. Extended abstract in Workshop on the Design of Dependable Critical Systems, Hamburg, 2009.

Mekacher Zouaghi, L., A. Wagner and E. Badreddin: Hybrid, recursive, nested monitoring of control systems using Petri nets and particle filters. Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010), 73-79, Chicago, June 28 - July 01, 2010.

Wagner A., L. Mekacher Zouaghi, F. Barth, C. Atkinson, E. Badreddin: A Unified Design Approach for Component-Based Dynamic Systems Monitoring. IEEE/SICE International Symposium on System Integration (SII), 2011, pp.1137-1142.

Mekacher Zouaghi, L., A. Alexopoulos, A. Wagner, E. Badreddin: Modified Particle Petri Nets for Hybrid Dynamical Systems Monitoring Under Environmental Uncertainties, IEEE/SICE International Symposium on System Integration (SII),2011, pp. 497-502.

Mekacher Zouaghi L, Alexopoulos A, Wagner A, Badreddin E.: Probabilistic Online-Generated Monitoring Models for Mobile Robot Navigation using Modified Petri nets, Tallinn, Estonia, 2011.

Mekacher Zouaghi L., Wagner A., Badreddin E., Monitoring of Hybrid Systems using Behavioural System Specification, accepted by 18th IFAC World Congress, Milano, Italy, 2011.

Mekacher Zouaghi, L., A. Wagner and E. Badreddin: Benutzerinteraktion beim Online-Monitoring der Navigation eines autonomen Rollstuhls. Proceedings of AUTOMED-Automatisierungstechnische Verfahren für die Medizin, Aachen, Germany, March 29-30, 2012.

Mekacher Zouaghi, L., A. Alexopolous, M. Koslowski, A. A. Kandil and E. Badreddin: An integrated distributed monitoring for mission-based systems: On the example of an autonomous unmanned helicopter. 6th IEEE International Conference on Intelligent Systems (IS), pp. 425-420, Sofia, Bulgaria, 6-8 Sept. 2012.

Alexopoulos, A., L. Mekacher Zouaghi and E. Badreddin: Associative memory for modified Petrinet based monitoring of mobile robot navigation. 1st International Conference on Innovative Engineering Systems (ICIES), Alexandria, Egypt, December, 7-9, 2012.

Leila Mekacher Zouaghi, Achim Wagner, Essameddin Badreddin: „Design of Fault-Tolerant and Dependable Assistance Systems with Degree of Autonomy Adaptation“ On The International Conference on Advanced Logistics and Transport (ICALT'2013) May 2013, in Sousse, Tunisia.

Nargess Sadaghzadeh, Leila Mekacher Zouaghi, Achim Wagner, Javad Poshtan, Essameddin Badreddin: “Cascaded Error Estimation and Compensation with Optical and Inertial Sensors Fusion in Robotic Surgery“. On the competitiveness through Maintenance and Assest Management (COMADEM 2013), June 2013, in Helsinki, Finland

Nargess Sadeghzadeh Nokhodberiz, Leila Mekacher Zouaghi, Achim Wagner, Eugen Nordheimer, Javad Poshtan and Essameddin Badreddin. Cascaded Kalman Filters for Error Estimation and Compensation of Inertial Navigation System Fusing Optical and Inertial Sensors. In Proceedings of the 5th International Conference on Mechatronics (ICOM 2013). Juli 2013.

Achim Wagner, Leila Mekacher Zouaghi, Florian Barth, Colin Atkinson, and Essameddin Badreddin: „ Behavior-Based Built-in Testing and Monitoring for Dependability Assessment of Dynamical Systems“. On the competitiveness through maintenance and Assest Management (COMADEM 2013), June 2013, in Helsinki, Finland.

Leila Mekacher Zouaghi, Alexander Alexopoulos, Achim Wagner, Essam Badreddin: Mission-based online generation of probabilistic monitoring models for mobile robot navigation using Petri nets. Robotics and Autonomous Systems 62(1): 61-67 (2014)

2 Theoretische Grundlagen der Systementwicklung und Überwachung

Im vorherigen Kapitel wurden die Problemstellungen sowie Ziele dieser Arbeit erläutert. In diesem Kapitel werden die Grundlagen für diese Arbeit vorgestellt. In Abschnitt 2.1 wird auf die komponentenbasierte Systementwicklung eingegangen. Hierbei wird, nach einer Vorstellung des Stands der Technik, hauptsächlich der Kobra-Ansatz angesprochen. Da eines der Ziele dieser Arbeit die Nutzung von formalen Modellen in Kobra ist, wird auf die formale Beschreibung der Komponenten und vor allem auf die Graphentheorie eingegangen. In Abschnitt 2.2 werden die Grundlagen für die Beschreibung von dynamischen hybriden Systemen vorgestellt. Auf die Überwachung und Fehlerdiagnose wird in Abschnitt 2.3 eingegangen, da zwischen den beiden Ansätzen eine enge Verbindung besteht. Ansätze mit hybriden Modellen werden in Abschnitt 2.4 diskutiert, um den Stand der Technik zu definieren und Defizite festzulegen. In Abschnitt 2.5 wird auf die notwendigen Grundlagen für die stochastische Zustandsschätzung und in Abschnitt 2.6 auf die Petri-Netze eingegangen. Beide Modelle stellen die Säulen des in dieser Arbeit entwickelten Verfahrens dar.

2.1 Komponentenbasierte Systementwicklung

Eine komponentenbasierte Software- bzw. Systementwicklung (engl., Component Based Development – CBD) entspricht einem Entwicklungsansatz auf Basis von Komponenten, die wiederverwendet werden können. Die Kernidee dieses Ansatzes ist der einfache Systementwurf, die Kapselung von Funktionalitäten in standardisierten Modulen und die Änderungseinfachheit durch Austausch bzw. Wiederverwendbarkeit dieser Module, für einen besseren Umgang mit der höheren Änderungsrate heutiger Systeme. Aus diesem Grund liegt das Hauptaugenmerk bei der Entwicklung dieser Komponenten auf der Verwendung wohldefinierter und einheitlicher Schnittstellen. Im Entwicklungsprozess ist es auch wichtig, dass bei auftretenden Fehlern die betroffenen Komponenten in einen vorherigen fehlerfreien stabilen Zustand wiederhergestellt werden ([Mat10]). Das Ziel bei dieser Entwicklung von wiederverwendbaren, einheitlichen, universell einsetzbaren Komponenten ist es, sichere Systeme zu entwickeln, die Zuverlässigkeit zu steigern, die System-Entwicklungszeit zu verkürzen, die Fehleranfälligkeit zu reduzieren und somit auch die Kosten stark zu senken ([Hei01]-[Dim05]).

2.1.1 Stand der Technik von Komponentenmodellen?

Für die Beschreibung der Komponenten gibt es verschiedene industrielle Ansätze von Komponentenmodellen, wie z. B. CORBA ([Omg06]), COM/DCOM ([Mic06a]) und Enterprise JavaBeans ([Mic06b]). Diese verwenden eigene Notationen zur Beschreibung der Schnittstellen einer Komponente. Für die Beschreibung der Semantik werden verschiedene Formalismen eingesetzt. Zu nennen sind beispielsweise Petri-Netze, Prozessalgebra, Prädikatenlogik, Temporale Logik, Object Constraint Language (OCL) und Unified Modeling Language (UML).

Es gibt eine Vielzahl von Ansätzen ([Ake04]), die Komponentenmodelle definieren und für Forschungszwecke nutzen (Abbildung 2.1.1). Das Vorhandensein einer großen Anzahl von Komponentenmodellen hat die Forscher motiviert, diese vorhandenen Komponentenmodelle nach verschiedenen Kriterien zu bewerten ([Crn07]-[KuK07]-[Her08]-[Kou09]-[Ari11]). In [KuK07] wurden Taxonomien von Komponentenmodellen auf die Komponentenzusammensetzung untersucht. Komponentensemantik und -syntax wurden auch in der Bewertung berücksichtigt. Insgesamt haben 13 Komponentenmodelle die Kriterien der Bewertung erfüllt. Aufgenommen wurden Komponentenmodelle, die einen ausgereiften Entwicklungsstand mit ausreichender Dokumentation erreicht haben. Diese sind: .NET, Acme-like, ADL, CCM, COM, EJB, Fraktal, JavaBeans, Koala, Kobra, PECOS, SOFA, UML 2.0 / UML und WebServices.

• AUTOSAR	• JavaBeans
• BIP	• MS COM
• BlueArX	• OpenCOM
• CCM	• OSGi
• COMDES II	• Palladio
• CompoNETS	• PECOS
• EJB	• Pin
• Fractal	• ProCom
• KOALA	• ROBOCOP
• Kobra	• RUBUS
• IEC 61131	• SaveCCM
• IEC 61499	• SOFA 2.0

Abbildung 2.1.1: Vielzahl von Ansätze für Komponentenmodelle

In einem weiteren Wettbewerb ([Rau08]) wurden Komponentenmodelle auf deren praktische Anwendung sowie deren Spezifikationstechniken bewertet und verglichen. Im Rahmen des Wettbewerbs wurde eine Textbeschreibung und Java Implementierung eines Handelssystems als „Common Component Modelling Example“ (CoCoME) zur Verfügung gestellt ([Her08]),

und jedes teilnehmende Team hat seinen Komponentenmodellierungsansatz verwendet, um das Beispiel zu modellieren. Danach wurden die jeweiligen Modellierungen bewertet. Von 19 Teams haben 13 ihre Komponentenmodelle erfolgreich eingesetzt. Diese sind: CoIn, Cowch, DisCComp, Focus / Autofocus, Fraktal, GCM / ProActive, Java / A, Klaper, Kobra, Palladio, SOFA, RCOS und Reich Services.

Die Wettbewerbe und Evaluationen zeigen, dass jede komponentenbasierte Methode sowohl Stärken als auch Schwächen aufweist. Keiner der Ansätze kann alle Modellierungsmöglichkeiten abdecken. Dies ist nachvollziehbar, denn diese Komponentenmodelle stellen den aktuellen Stand der Forschung dar ([Mat10]) und lassen sich in semi-formale und formale Methoden klassifizieren:

- a) Formale Ansätze: Diese Gruppe bietet formale Modelle an und ermöglicht somit die Verifikation sowie die Validierung der funktionalen Korrektheit der Spezifikation. Hierfür werden verschiedene Semantiken eingesetzt, z. B. Zustandssemantik ([Che07]), algebraische Semantik ([Bro07]) oder Objektsemantik ([App07]). Weitere formale Ansätze erlauben die Simulation und Vorhersage einiger Systemeigenschaften hinsichtlich gewünschter Qualität des Verhaltens, wie Leistungsfähigkeit und Zuverlässigkeit ([Kro07], [Gra07], [Bul07], [Bur07] und [Bec09]).
- b) Semi-formale Ansätze: Diese Ansätze definieren eigene semi-formale Modelle, als Erweiterungen der UML-Semantik, z. B. Kobra ([Atk07]) und Rich Services ([Dem07]). Sie integrieren verschiedene Konzepte und bieten somit einige Vorteile, worüber andere formale Ansätze nicht verfügen. Kobra integriert beispielsweise folgende Konzepte:
 - Modellgetriebenes Engineering
 - Komponentenbasiertes Engineering
 - Objektorientiertes Engineering
 - Produktlinien Engineering
 - Rekursive Top-down Zerlegung
 - Qualitätssicherung
 - Software Entwurfsmuster

Die Integration vom Built-in Test-Konzept macht die Kobra-Methode interessant für die Entwicklung von sicherheitskritischen Systemen, welche eine Verifikation der Funktionalitäten erfordern. Eine Erweiterung dieser Methode auf formalen Modellen sollte

weitere Vorteile hinsichtlich der automatischen Verifikation sowie der Einbettung in Echtzeitsystemen mit sich bringen. Ein Link zur Überwachung wird für Kobra durch diese Arbeit bereitgestellt.

2.1.2 Der Ansatz Kobra

Kobra wurde mit zwei grundlegenden Zielsetzungen entwickelt ([Atk03b]). Das erste Ziel ist die Unterstützung eines modellgetriebenen Ansatzes zur komponentenbasierten Software-Entwicklung. Hierfür ist ein Modell der Komponenten und ihrer Beziehungen erforderlich, z. B. mittels graphischer Diagramme, wie die vereinheitlichte Modellierungssprache UML (engl., Unified Modeling Language). Mit diesem Ansatz werden die Komponenten und die komponentenbasierten Architekturen, unabhängig von deren Implementierung, erfasst, was eine Beschreibung auf hoher Abstraktionsebene erlaubt. Dies macht die Architektur viel stabiler in Bezug auf Veränderungen. Das zweite Ziel der Kobra-Entwicklung ist die rekursive Modellierung, in der Komponenten bis in beliebige Tiefen geschachtelt werden können. Der Vorteil dieser rekursiven Komponentenmodellierung ist, dass alle Komponenten unabhängig von ihrer Größe oder ihrer Stellung in der Kompositionshierarchie gleich behandelt werden. Somit können dieselben Modellierungs- und Entwicklungsgrundsätze gleichmäßig angewendet werden. Kobra macht keine Unterscheidung zwischen „System“ und „Komponente“; die Komponentenentwicklung entspricht der Komponentenzusammensetzung, und somit verhält sich eine Komponente ähnlich wie ein System.

Nach Kobra wird jede Komponente in einem System durch ihre Spezifikation und Realisierung beschrieben, und zwar in drei Sichtweisen: eine funktionale, eine strukturelle und eine verhaltensbasierte Sichtweise (Abbildung 2.1.2). Die Spezifikation beschreibt die Außensicht der Komponente. Dabei werden das Verhalten, die Schnittstellen und die Struktur spezifiziert, also die nach außen sichtbaren Eigenschaften einer Komponente. Die Realisierung andererseits beschreibt, wie eine Komponente ihre durch die Spezifikation festgelegten Eigenschaften bzw. gewünschten Anforderungen realisiert. Für jede Phase werden die Sichtweisen durch bestimmte Diagramme bzw. Techniken beschrieben. Die Komponentenspezifikation besteht aus einem Strukturmodell zur Beschreibung der Interaktion, einem Verhaltensmodell zur Beschreibung der äußerlich sichtbaren Zustände einer Komponente und einem funktionalen Modell zur Beschreibung der Operationen ([Atk03a]). Eine Komponentenrealisierung besteht aus einem Strukturmodell, das das Spezifikationsstrukturmodell durch Funktionalitäten der Komponente erweitert, einem Interaktionsmodell zur Beschreibung der Operationenrealisierung sowie der Interaktionen mit

anderen Objekten und Komponenten sowie einem Aktivitätsmodell zur Beschreibung der Algorithmen für die Implementierung der Operation.

Specification

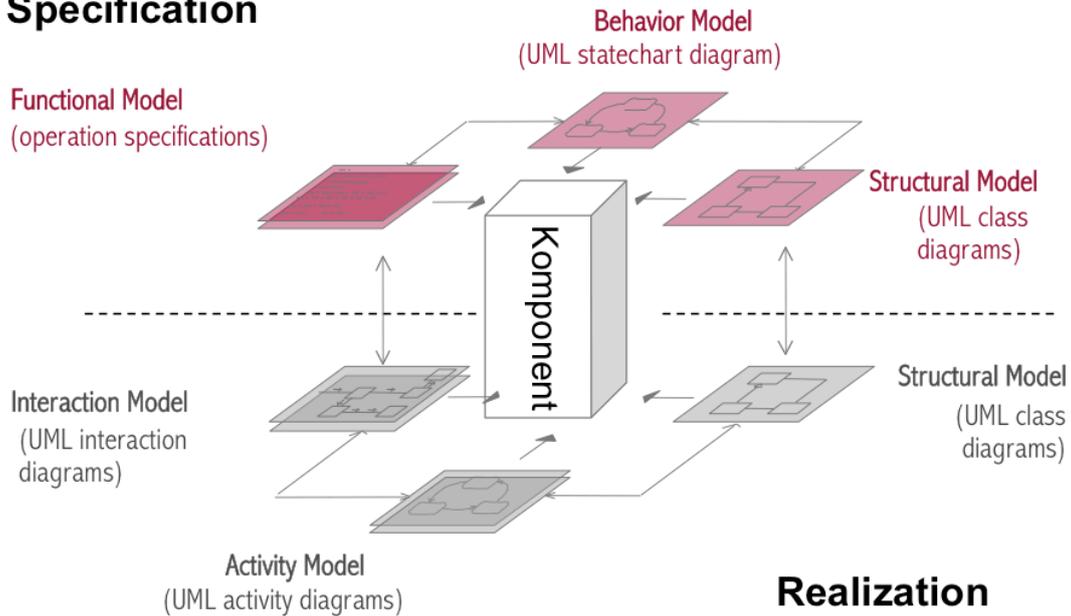


Abbildung 2.1.2: Modellbasierte Komponentenschreibung nach Kobra ([Atk03a])

- Komponentenzerlegung in Kobra:

Die komponentenbasierte Entwicklung ermöglicht die Beherrschung der zunehmenden Komplexität durch Unterteilen des Gesamtsystems in unabhängige kleinere Einheiten. Sie erhöht damit die Flexibilität und Stabilität des Gesamtsystems. Abbildung 2.1.3 zeigt das Konzept von Kobra für die Zerlegung des realen Systems in Komponenten.

In Abbildung 2.1.4 handelt es sich um die Beschreibung eines Steuerungssystems in einem Fahrzeug. Dieses lässt sich in drei Regelungssysteme zerlegen: das Positionsregelungs-, Geschwindigkeitsregelungs- und Antriebsregelungssystem. Um eine bestimmte Position zu erreichen, werden die entsprechenden Geschwindigkeitsvorgaben für das Fahrzeug berechnet. Daraus werden die Geschwindigkeiten der angetriebenen Räder bzw. der Motoren als Pulsweitenmodulationen (engl., Pulse Width Modulation, PWM), die die Bewegung bestimmen, ermittelt.

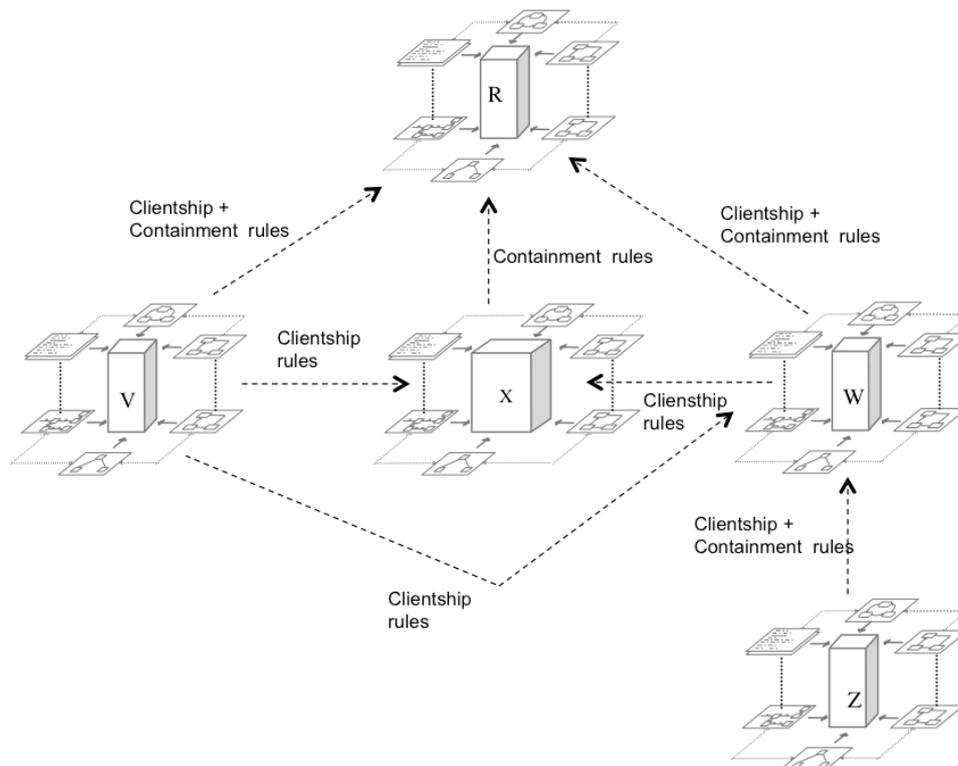


Abbildung 2.1.3: Konzept der Komponentenerlegung in Kobra [Atk02a]

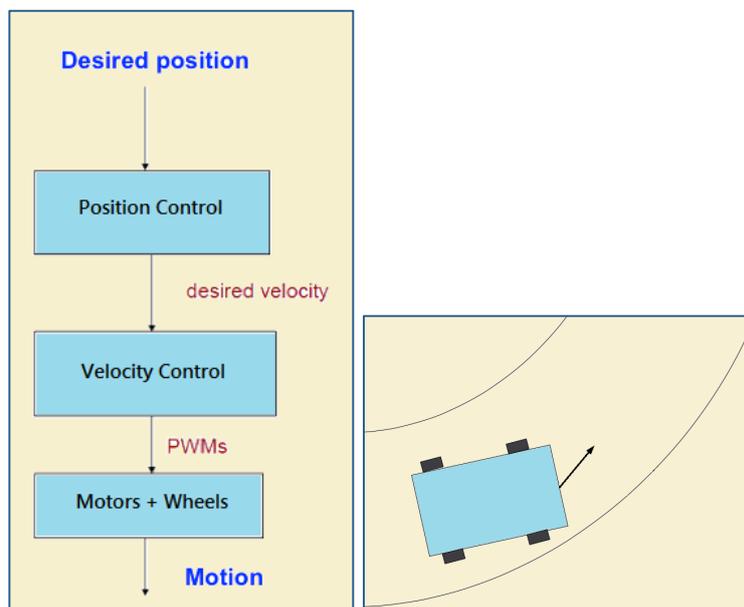


Abbildung 2.1.4: Zerlegung eines Fahrzeugsteuerungssystems in Komponenten

Abbildung 2.1.5 zeigt die strukturelle Spezifikation der Komponente „Geschwindigkeitsregelung“ in der Hierarchie. Es wird nur die äußere Sicht dieser Komponente beschrieben, also welche anderen Komponenten sie benötigt. Auf Subkomponenten wird nicht eingegangen. Die Komponente, deren Spezifikation dargestellt wird, wird durch den Stereotyp <<subject>> gekennzeichnet.

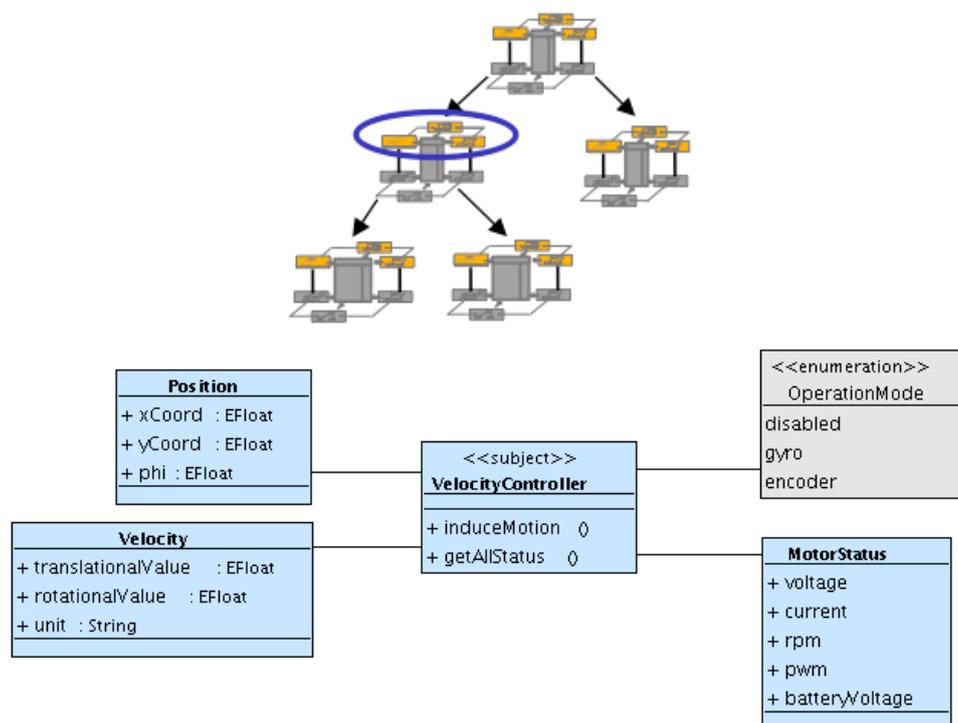


Abbildung 2.1.5: Strukturelle Spezifikation der Komponente „Geschwindigkeitsregelung“

Abbildung 2.1.6 zeigt das Klassendiagramm der Realisierung unseres Fallbeispiels. Dieses ergänzt das Modell der Spezifikation um die zur Realisierung benötigten Elemente.

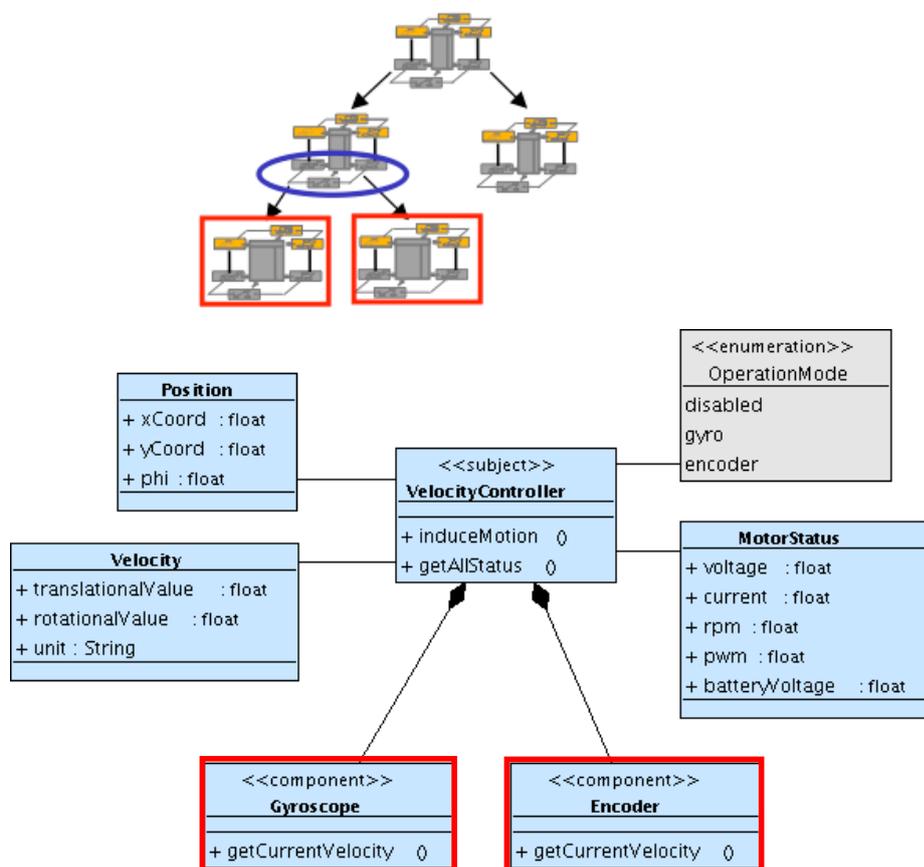


Abbildung 2.1.6: Strukturelle Realisierung der Komponente „Geschwindigkeitsregelung“

Im Fall der Geschwindigkeitsregelung werden beispielsweise zusätzlich die Komponenten „Radencoder“ und „Gyroskop“ benötigt. Es handelt sich hier um zwei Sensorsysteme zur Geschwindigkeitsmessung. Zur Unterscheidung der Klassen dieser Unterkomponenten von einfachen Klassen der anderen Komponenten werden diese mit dem Stereotyp <<component>> gekennzeichnet. Werden diese Komponenten selbstentwickelt, so muss wiederum, im Rahmen des rekursiven Entwicklungsprozesses für diese Komponenten, eine Spezifikation und eine Realisierung erfolgen.

Im Weiteren werden, anhand eines Beispiels für die Temperaturregelung in den Räumen eines Hauses, unterschiedliche Kobra-Diagramme für die verschiedenen Sichten einer Komponente veranschaulicht. Abbildung 2.1.7 zeigt die strukturelle Spezifikation der Komponente „Room“, die wiederum aus einer Komponente „Thermostat“ besteht.

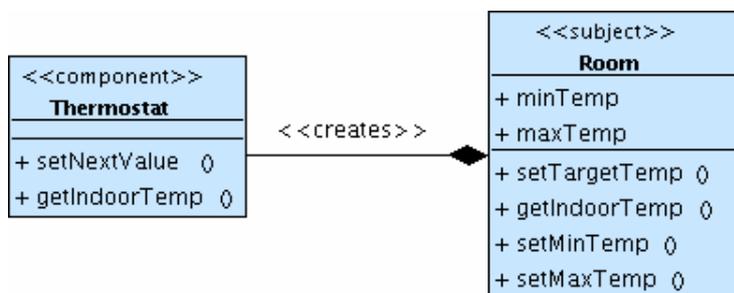


Abbildung 2.1.7: Strukturelle Spezifikation der Komponente „Room“

Der Thermostat hat die Aufgabe, die Temperatur in einem Raum zwischen einem minimalen und einem maximalen Wert zu halten. Er bekommt die Information über die Innentemperatur im Raum „getIndoorTemp()“ und setzt einen Wert „setNextValue()“ innerhalb des spezifizierten Temperaturintervalls. Abbildung 2.1.8 zeigt die funktionale Spezifikation der Komponente „Thermostat“, also die Dienste, die sie liefern soll.

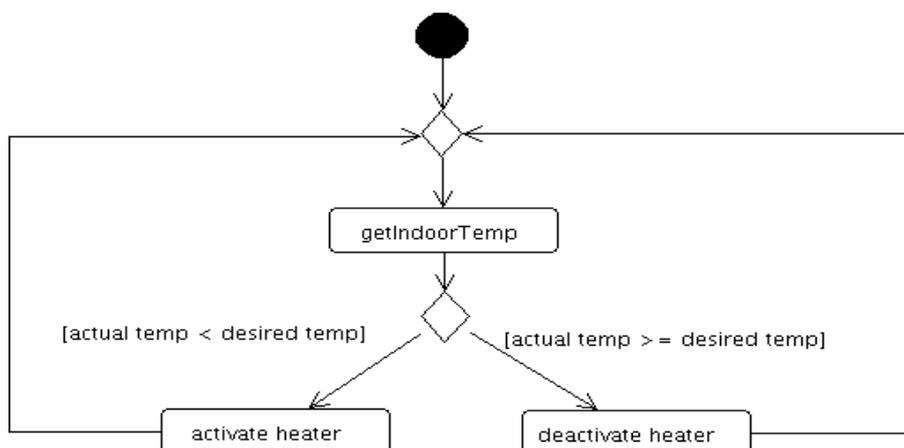
Name	setNextValue
Description	Sets the temperature value that has to be maintained. The Thermostat stores only this single temperature value.
Receives	tempValue: number
Precondition	The given value must be within the range specified by <i>maxTemp</i> and <i>minTemp</i> .

Name	getIndoorTemp
Description	Returns the temperature prevalent at the time when the method is called.
Returns	tempValue: number

Abbildung 2.1.8: Funktionale Spezifikation der Komponente „Thermostat“

Die Prozesse für die Temperaturmessung bzw. das Heizen und Kühlen, um die Temperatur des gewünschten Wertes einzuhalten, werden durch ein Verhaltensmodell spezifiziert bzw. realisiert (Abbildung 2.1.9).

Verhaltensspezifikation:



Verhaltensrealisierung:

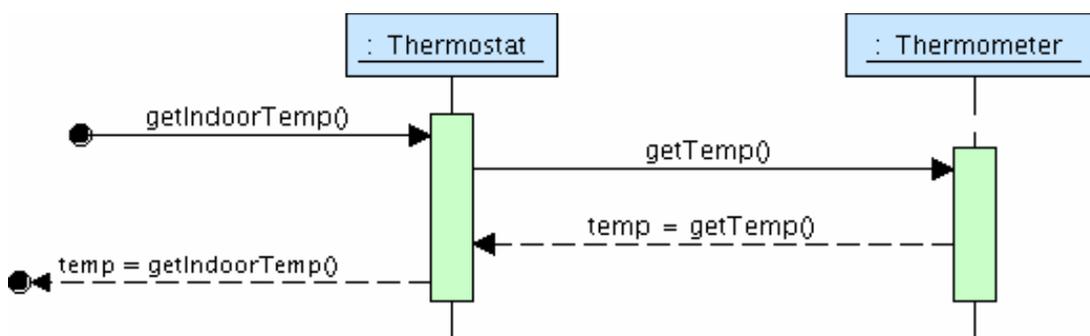


Abbildung 2.1.9: Statechart und Aktivitätsdiagramm für die Verhaltensspezifikation und -realisierung

- Testschablonen („TestSheet“):

In einem TestSheet ([Fel12]) werden die erwarteten Funktionalitäten einer Komponente durch die Definition der Beziehungen zwischen den Eingangs- und Ausgangsparametern in einer Tabellenkalkulation festgelegt. Tabelle 2.1.1 zeigt ein einfaches Beispiel eines TestSheets zum Testen eines Daten Stack. Jede Zeile steht für einen Operationsaufruf. Die Spalten A und B enthalten den Service zum Aufruf und die Operation, die durch den Test aufgerufen werden soll. Die Spalte C steht für die Eingangsvariablen und Spalte D enthält die erwarteten Ergebnisse. Neben der System-Initialisierung enthält der Überwachung-TestSheet eine Klasse zum Daten-Provider und eine Klasse für die Zuverlässigkeitsberechnung. Weitere Einzelheiten über TestSheets sind in [Atk08] zu finden.

	A	B	C	D
1	CalculationHelper	init		
2	DataProvider	init		
3	D2	isDataAvailable		
4	D2	getReferenceData		
5	D2	getRealisedData		C5-C6
6	→ 7 / 1 2 3			
7	[C3] → 7 / 4 5	[!C3] → 8		
8				
9	<relevant measures calculated from column D>			

	A	B	C	D
1	DepTestSheet	test	true	<returned relevant measures>
2	DepenabilityCalculator	compute	D1	

Tabelle 2.1.1: TestSheet [Wag13]

2.1.3 Formale Beschreibung der Komponentenverhalten

Die Wiederverwendung einer Komponente erfordert eine ausreichende Dokumentation ihrer Funktionalitäten und Schnittstellen. Diese dokumentierten Informationen werden in einer *Komponentenspezifikation*, die funktionale und nicht-funktionale Anforderungen beinhaltet, festgehalten. Die Nutzung der natürlichen Sprache für die Spezifikation führt in vielen Fällen zu Missverständnissen. Dieses Problem tritt bei der Nutzung formaler Spezifikations Sprachen nicht auf, weil sie eindeutig sind, auf mathematischen Grundlagen basieren und eine streng definierte Semantik sowie Syntax besitzen. Formale Methoden ermöglichen somit eine eindeutige Interpretation der Spezifikation sowie eine Überprüfung der Modelle und Komponentenschnittstellen. Unnötige Anforderungen lassen sich frühzeitig erkennen und entfernen, z. B. dass es keine mögliche Folge von Systemaktionen gibt, die zu einem unsicheren Zustand führen könnte.

Die Petri-Netze zeigen allgemein viele Vorteile gegenüber anderen Ansätzen und werden deswegen in der Praxis vielfach benutzt. Denn zusätzlich zu der intuitiven graphischen Darstellung und zu der Fähigkeit, nebenläufige Prozesse zu modellieren, bietet die Verwendung der Petri-Netze eine große Unterstützung beim Entwurf ([Les88]-[Fre98]). Sie erlauben eine frühzeitige Systemanalyse, z. B.:

- Ist das entworfene System überhaupt realisierbar? (→ Beschränktheit des Netzes)
- Kann das entworfene System gewisse erwünschte bzw. unerwünschte Zustände einnehmen? (→ Erreichbarkeit einer bestimmten Markierung im Netz-Modell)
- Kann das entworfene System in einen Verklemmungszustand geraten? (→ Erreichbarkeit eines Deadlocks im Netz-Modell).

Petri-Netze in ihren verschiedenen Arten erlauben die Modellierung auf unterschiedlichen Abstraktionsebenen und die Verifikation der Modelle anhand von Software-Werkzeugen, die die mathematischen Regeln aus der Petri-Netz-Theorie implementieren.

Eine Literaturrecherche in den Bereichen „formale Methoden für den Systementwurf“ und „Verifikation“ zeigt fünf Klassen von Ansätzen [Bar92]:

- Modellbasierte Ansätze: Sie bieten zwar eine explizite und abstrakte Definition von Systemzuständen und Transitionen, aber keine Darstellung von Nebenläufigkeit, z. B. *Z, VDM und Methode B*.
- Algebraische Ansätze: Sie bieten eine implizite Definition der Operationen und die mathematische Formulierung der Beziehungen zwischen den Objekten, jedoch keine Definition von Zuständen und keine Darstellung von Nebenläufigkeit, z. B. *OBJ, PLUSS*.
- Prozessalgebras: Ein explizites Modell für nebenläufige Prozesse, mit der Darstellung des Verhaltens durch Bedingungen über die zulässige Kommunikation zwischen den Prozessen, z. B. *CSP, CCS und OCCAM*.
- Logikbasierte Ansätze: Sie nutzen Logiken, um die Eigenschaften eines Systems zu beschreiben, z. B. *Temporale Logik, Intervall Logiken, Propositionale Logik und Prädikatenlogik*.
- Netzbasierter Ansätze: Sie beinhalten die Darstellung des Verhaltens durch ein Netzwerk. Die Daten fließen unter Bedingungen von einem Knoten zum anderen und Nebenläufigkeit ist erlaubt, z. B. *Petri-Netze und Prädikat Transitions Netze*.

Mit diesen Ansätzen werden die Beziehungen zwischen den Komponenten einheitlich und formal beschrieben. Diese Beschreibungen stellen den Ausgangspunkt der Modellierung dar.

2.1.4 Graphentheorie in Software-/Systemengineering

Die Graphentheorie ([Tur04]) findet in verschiedenen Bereichen Einsatz und stellt ein anschauliches und klares Beschreibungsmittel dar. Graphen werden in der Systemtheorie ([Rop12]) zur Modellbildung von Systemen und in der Softwaretechnik zur Architekturbeschreibung eingesetzt ([Beh08]). Die Objekte werden als Knoten und die Verbindungen zwischen den Objekten als Kanten dargestellt. Somit kann jedes komponentenbasierte Softwaresystem als Graph dargestellt werden.

Definition 2.1.1: Ein **Graph** ist ein Paar $G = (V, E)$, wobei V eine endliche Menge von Knoten (vertex) und E eine endliche Menge von Kanten (edge) darstellt. Es gilt $E \subseteq V \times V$ und $V \cap E = \emptyset$. Ein Graph G kann gerichtet oder ungerichtet sein. Ein ungerichteter Graph ist ein Graph, bei dem die Kanten ungerichtet sind.

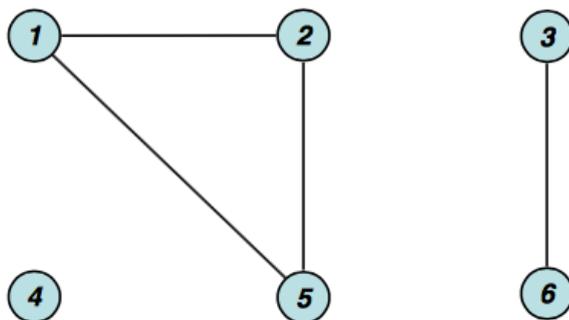


Abbildung 2.1.10: Ungerichteter Graph

Beispiel: Der Graph in Abbildung 2.1.10 beinhaltet die Knotenmenge $V = \{1,2,3,4,5,6\}$ und die Kantenmenge $E = \{(1,2), (1,5), (5,2), (3,6)\}$.

Definition 2.1.2: Ein **gerichteter Graph** ist ein Graph, der zwei Abbildungen enthält. Eine Abbildung $init: E \rightarrow V$ bzw. eine Abbildung $end: E \rightarrow V$, die jeder Kante e einen Anfangsknoten bzw. einen Endknoten zuordnet.

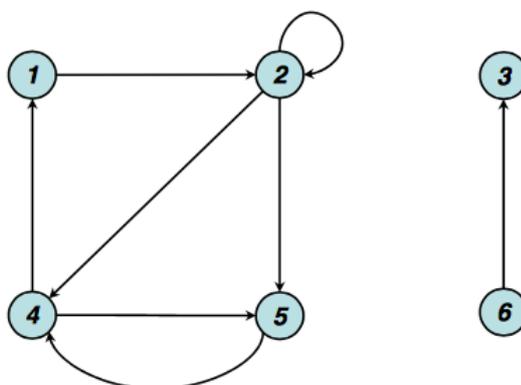


Abbildung 2.1.11: Gerichteter Graph

Beispiel: Der Graph in Abbildung 2.1.11 beinhaltet die Knotenmenge $V = \{1,2,3,4,5,6\}$ und die Kantenmenge $E = \{(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)\}$

Definition 2.1.3: Die Inzidenzmatrix $I = (i_{jk})_{n \times m}$ eines gerichteten Graphen G mit der Knotenmenge $V = \{v_1, \dots, v_n\}$ und der Kantenmenge $E = \{e_1, \dots, e_m\}$ wird wie folgt definiert:

$$i_{jk} = \begin{cases} 1 & \text{wenn } v_j \text{ der Ursprung von } v_k \text{ ist} \\ -1 & \text{wenn } v_j \text{ das Ende von } v_k \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Diese formale Notation ist die Grundlage von verschiedenen, in der Literatur bekannten, Graphenmodellen, wie etwa Petri-Netzen oder endliche Zustandsautomaten. Diese werden in Zusammenhang mit der Systemmodellierung im nächsten Abschnitt vorgestellt.

2.2 Grundlagen dynamischer Systemmodellierung

Um Systeme klassifizieren zu können soll eine geeignete formale Systembeschreibung, also ein geeignetes Modell des Prozesses, gefunden werden. Wenn von Systemklassifikation und Systemklassen die Rede ist, sind damit die Modellklassifikation und Modellklassen gemeint. Abbildung 2.2.1 zeigt die verschiedenen Klassen von dynamischen Systemen.

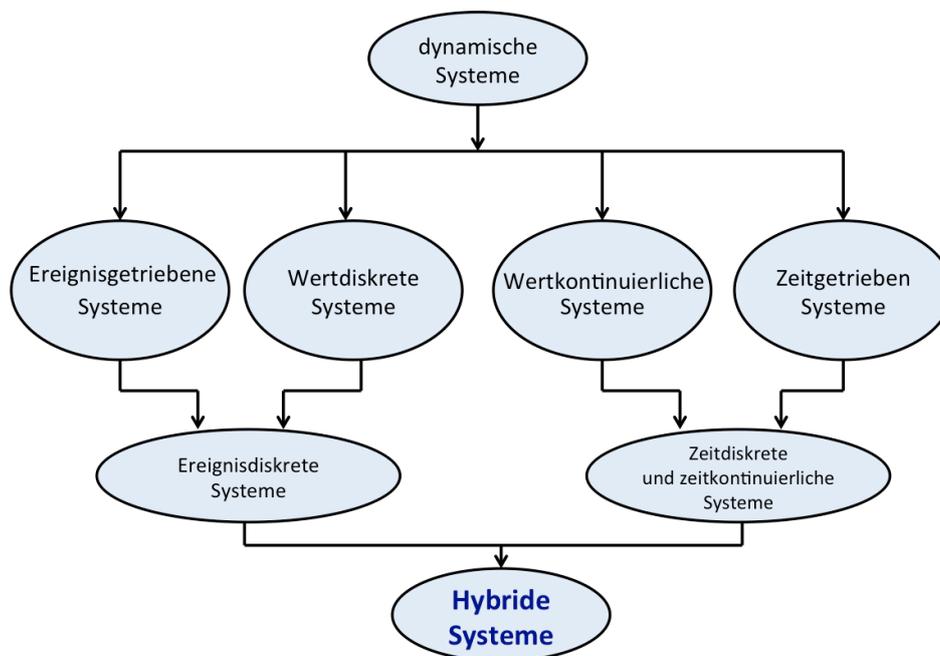


Abbildung 2.2.1: Klassen der dynamischen Systeme

Wir beschränken uns auf dynamische Systeme, da sich heutige Systeme meist in quantifizierbaren Größen messen lassen (Temperatur, Winkel, Geschwindigkeit etc.) In diesem Abschnitt wird zunächst zwischen statischen und dynamischen Systemen unterschieden. Anhand der verschiedenen Arten von Zustandsübergängen werden die einzelnen Systemklassen und deren Beschreibungsmethoden vorgestellt, um den *hybriden Formalismus* einzuführen.

2.2.1 Dynamische Systembeschreibung

Das Zeitverhalten eines Systems ergibt sich aus der zeitlichen Änderung seiner Zustandsvariablen. Sind alle Zustandsvariablen konstant, so handelt es sich um ein statisches System. Im anderen Fall liegt ein dynamisches System vor. Im Gegensatz zu statischen Systemen ist das Wesen von dynamischen Systemen dadurch geprägt, dass die Vorgeschichte des Systems und die aktuellen Eingangsgrößen den weiteren Verlauf der Systemgrößen beeinflussen. Die momentanen Werte der Ausgangsgrößen sind damit nicht nur von den momentanen Werten der Eingangsgrößen, sondern auch vom bisherigen Verlauf der Systemgrößen – so genannte Systemzustände - über die Zeit abhängig. Zur geeigneten Beschreibung solcher dynamischer Systeme werden entsprechende Formalismen benötigt. In Anlehnung an die Definition in [Wil91] wird hier ein dynamisches System folgendermaßen beschrieben:

Definition 2.2.1 ([Wil91]): Ein dynamisches System Σ ist durch ein Tripel $\Sigma = (T, W, B)$ dargestellt (Abbildung 2.2.2). Dabei ist T die Menge der Zeitpunkte aus \mathfrak{R}^+ (kontinuierlich) oder \mathbb{N}^+ (diskret), die die Zeitachse bilden. W ist der Signalraum, und $B \subseteq W^T$ ist das Systemverhalten.

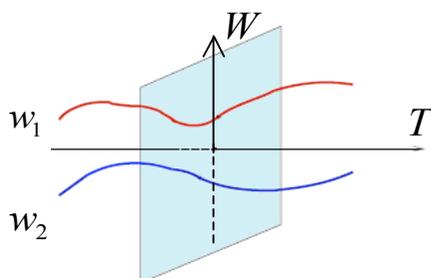


Abbildung 2.2.2: Signalraum eines dynamischen Systems

Ein dynamisches hybrides System ist eine gemischte Struktur aus kontinuierlichen und diskreten Teilsystemen mit kontinuierlich-diskreter Dynamik. Im weiteren Verlauf der Arbeit ist $W = U \times Y$, wobei U die Menge der Eingangssignale und Y ist die Menge der Ausgangssignale ist. Da diese Arbeit hauptsächlich hybride Systeme behandelt, die eine Verallgemeinerung der nichtlinearen Systeme darstellen, werden die Zustandsraummodelle für die Modellierung eine große Rolle spielen. Die Zustandsraumdarstellung (Abbildung 2.2.3) bzw. die (Eingang/Zustand/Ausgang)-Darstellung von Σ ist durch [Wil02]

$$\Sigma_s = (T, W, X, B_s) \quad (2.2.1)$$

repräsentiert, in der das Verhalten B_s die Zustandsaxiome erfüllt, und X bildet damit die Menge der zulässigen Zustände des Systems. Das Verhalten von Σ ist dann als

$$B = \{w \mid \exists x \text{ so dass } (w, x) \in B_s\}, \quad (2.2.2)$$

definiert.

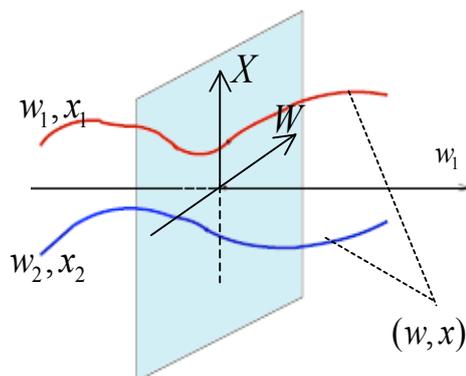


Abbildung 2.2.3: Zustandsraum eines dynamischen Systems

2.2.2 Systemdynamik und Zustandsübergänge

Das interne Verhalten bzw. die Dynamik eines Systems werden durch die Beschreibung der Zustandsübergänge bestimmt. Der Zustandsübergang spielt im Rahmen dieser Arbeit eine wichtige Rolle. Aus diesem Grund wird die Definition der Zustandsgröße und des Zustandsübergangs vorgestellt.

Definition Zustandsgröße: Zustandsgrößen sind Systemgrößen (Speichergrößen), die zu jedem Zeitpunkt und bei Kenntnis des Zeitverlaufes der Eingangsgrößen die Ermittlung des zukünftigen Systemverhaltens ermöglichen. Die Zustandsgrößen geben die Koordinaten des Verhaltensraumes eines Systems an. Die Zahl der Zustandsgrößen gibt die Dimension des Systems und die Zahl der beschreibenden Differentialgleichungen bzw. Differenzgleichungen an.

Definition Zustandsübergang: Zustandsübergänge beschreiben die Änderung des Zustands eines Systems bzw. Teilsystems. Bei kontinuierlichen Änderungen der Zustandsvariablen über die Zeit spricht man von einem kontinuierlichen System. Treten zu bestimmten Zeitpunkten Sprünge der Parameterwerte auf, d. h. diskontinuierliche Zustandsübergänge, spricht man von einem zeitdiskreten System. Ändert sich der Zustand aufgrund eines Ereignisses, spricht man von einem ereignisdiskreten Zustand. Im Folgenden werden diese verschiedenen Systemklassen eingeführt.

Ein zeitkontinuierliches System ist ein dynamisches System, bei dem die Zeitbasis T durch die Menge der reellen Zahlen \mathfrak{R}^+ dargestellt wird, wobei auch hier die Mengen X , U und Y jeweils Teilmengen von reellen Vektorräumen sind. Die Zustandsraummodelle ermöglichen die Beschreibung der Dynamik eines zeitkontinuierlichen Systems Σ_t durch Differentialgleichungen:

$$\Sigma_t : \begin{cases} \dot{x}_C(t) = f(x_C(t), u_C(t)), x_C(0) = x_0 \\ y_C(t) = g(x_C(t), u_C(t)) \end{cases} \quad (2.2.3)$$

Als Beschreibungstechnik kommen mathematische Ausdrucksformen zum Einsatz, wie beispielsweise algebraische Gleichungen, Differenzialgleichungen, Blockdiagramme, Übertragungsfunktionen und Bondgraphen ([Pay61]).

Ein zeitdiskretes System ist ein dynamisches System, bei dem die Zeitbasis T durch die Menge der ganzen Zahlen Z^+ dargestellt wird. Die Mengen X , U und Y sind jeweils Teilmengen von reellen Vektorräumen. In der Zustandsraumdarstellung ist die Dynamik eines zeitdiskreten Systems Σ_k durch Differenzengleichungen beschreibbar:

$$\Sigma_k : \begin{cases} x_{C,k+1} = f(x_{C,k}, u_{C,k}), x_{C,k=0} = x_0 \\ y_{C,k} = g(x_{C,k}, u_{C,k}) \end{cases} \quad (2.2.4)$$

Ein ereignisdiskretes System ist definiert durch eine Menge von Zuständen, die sich zu beliebigen, nicht getakteten Zeitpunkten, durch das Eintreten von Ereignissen ändern können. Das Modell verwendet die diskrete Zeitachse, auf der durch den Zähler k die Zustandsübergänge durchnummeriert sind. Es beschreibt, in welchen Nachfolgezustand $x_D(k+1)$ das System übergeht, wenn es zur Zeit k im Zustand $x_D(k)$ ist und die Eingabe $u_D(k)$ erhält. Weniger abstrakt gefasst, könnte das Überschreiten eines Schwellwertes als Ereignis definiert werden. Das Modell beschreibt auch, welche Ausgabe $y_D(k)$ in dieser Situation durch das System erzeugt wird:

$$\Sigma_D : \begin{cases} x_D(k+1) = F(x_D(k), u_D(k)), x_D(0) = x_0 \\ y_D(k) = G(x_D(k), u_D(k)) \end{cases} \quad (2.2.5)$$

Die gängigen Techniken zur Modellierung solcher Systeme sind endliche Automaten (auch Zustandsmaschine genannt), Petri-Netze ([Pet62]) und Statecharts ([Har87]).

Ein hybrides System entsteht aus der Kombination von ereignisdiskreten und zeitdiskreten Systemen oder aus der Kombination von ereignisdiskreten und zeitkontinuierlichen Systemen. Aufgrund der nicht-trivialen Interaktionen zwischen den kontinuierlichen Entwicklungsverläufen des Verhaltens und den diskreten Übergängen (Abbildung 2.2.4), ist es nicht möglich, diese Systeme entweder als rein kontinuierliche Systemvariablen oder als rein diskrete Ereignissysteme zu betrachten. Es ist zwangsläufig wichtig, dass die Kombination von beiden Teilen der Dynamik behandelt wird.

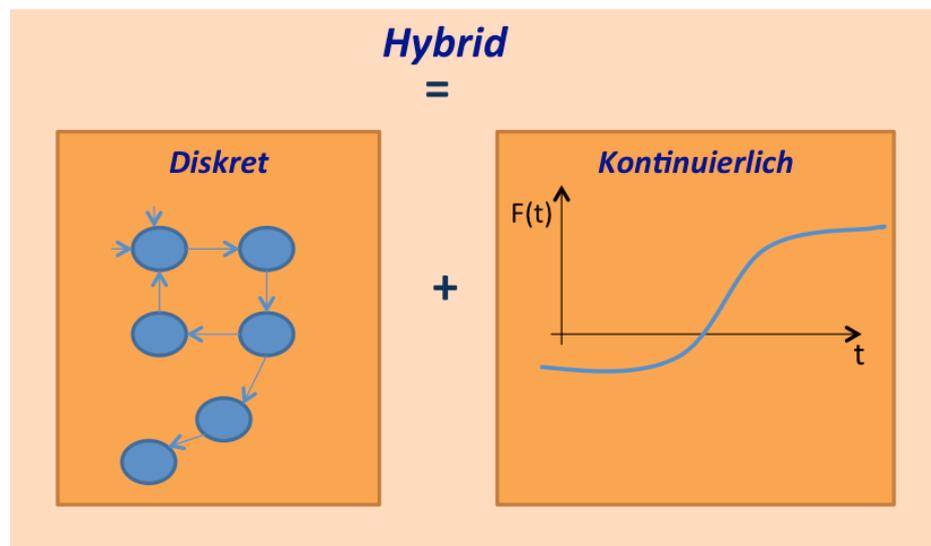


Abbildung 2.2.4: Hybride Systeme als Kombination der kontinuierlichen und diskreten Dynamik

Über mehrere Jahre ([Bra95]-[Bem99]-[Nen01]-[Sch01]) hat man festgestellt, dass die klassischen Modellierungsmethoden nicht direkt bei hybriden Systemen angewendet werden können. Einerseits abstrahieren die ereignisdiskreten Methoden das kontinuierliche Verhalten eines hybriden Systems zu diskreten Ereignissen und ignorieren damit die kontinuierliche Dynamik des Systems. Auf der anderen Seite behandeln die Methoden kontinuierlicher Systeme kein Umschalten zwischen den Modi, und somit modellieren sie nicht die diskrete Dynamik des hybriden Systems. Es gab in anderen Arbeiten Bemühungen, diese Forschungslücke zu überbrücken. Somit sind zwei Hauptkategorien von Modellierungsansätzen entstanden:

- Ansätze, die auf einer Erweiterung der Modellierungstechniken für kontinuierliche Systeme auf die diskrete Dynamik basieren, wie z. B. der Formalismus der gemischten Bondgraphen ([Dai07]- [Roy10]).
- Ansätze, die auf einer Erweiterung der Modellierungstechniken für ereignisdiskrete Systeme, wie z. B. hybride Petri-Netze oder hybride Automaten, basieren ([All98], ([Lyg03]).

2.2.3 Hybride Modelle

Modelle werden verwendet, um ein reales Systemverhalten nachzubilden und zu simulieren. Für die drei Modellklassen (kontinuierlich, diskret und hybrid) existieren folgende Darstellungsformen:

- *Ein mathematisches Modell:* Die Systemvariablen werden durch mathematische Funktionen, wie Differentialgleichungen oder Zustandsübergangswahrscheinlichkeiten, analytisch beschrieben.
- *Ein graphisches Modell:* Die Variablen werden durch Symbole dargestellt, z. B. wird in einem Petri-Netz der Systemzustand durch eine markierte oder nicht markierte Stelle beschrieben, und die Funktionen werden durch Aktivierungs-Schalt-Regeln realisiert. In einem Automaten werden die Variablen durch Ereignisse und gerichtete Kanten dargestellt.

Für die Modellierung der nicht-trivialen Wechselwirkungen zwischen beiden Teilen der Dynamik wurden meistens folgende werkzeuggestützte Modelle verwendet:

Hybride Automaten

Ein hybrider Automat besteht aus Zuständen und Transitionen, jedoch wird der Zustandsraum um kontinuierliche Variablen erweitert (Abbildung 2.2.5). Die Variablen in den einzelnen Zuständen werden mit Hilfe von Differenzialgleichungen ausgedrückt ([Hen96]). Zeitautomaten bilden eine Erweiterung um die Zeitkomponente ([Di194]).

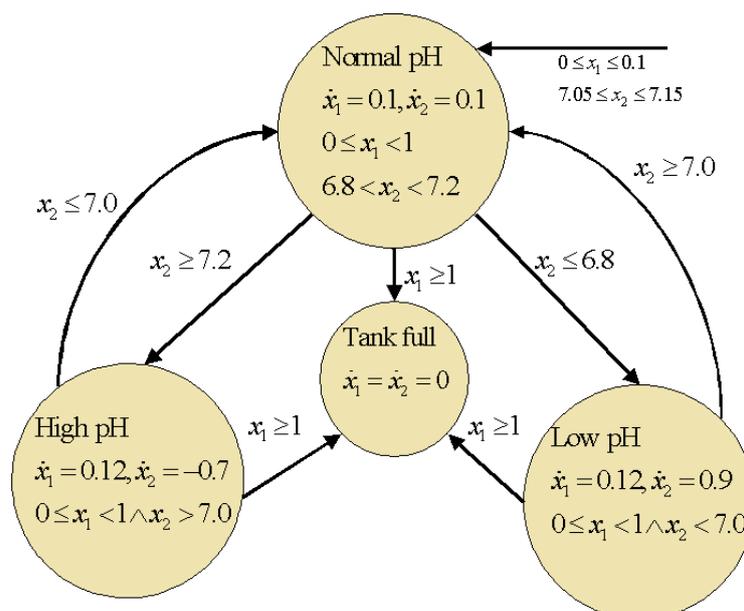


Abbildung 2.2.5: Hybride Automaten

Hybride Statecharts

Statecharts stellen eine Weiterentwicklung der endlichen Automaten dar ([Har87]). Der Statechart-Formalismus behebt zahlreiche Schwachpunkte der Automaten. Sie ermöglichen unter anderem das Schachteln von Zuständen, Zusammenfassen von komplexen Transitionen und die Darstellung der Nebenläufigkeiten. Die Erweiterung der gewöhnlichen Statecharts um Differenzialgleichungen ([Kes92]) hat es ermöglicht, kontinuierliche-diskrete Verhalten zu beschreiben (Abbildung 2.2.6).

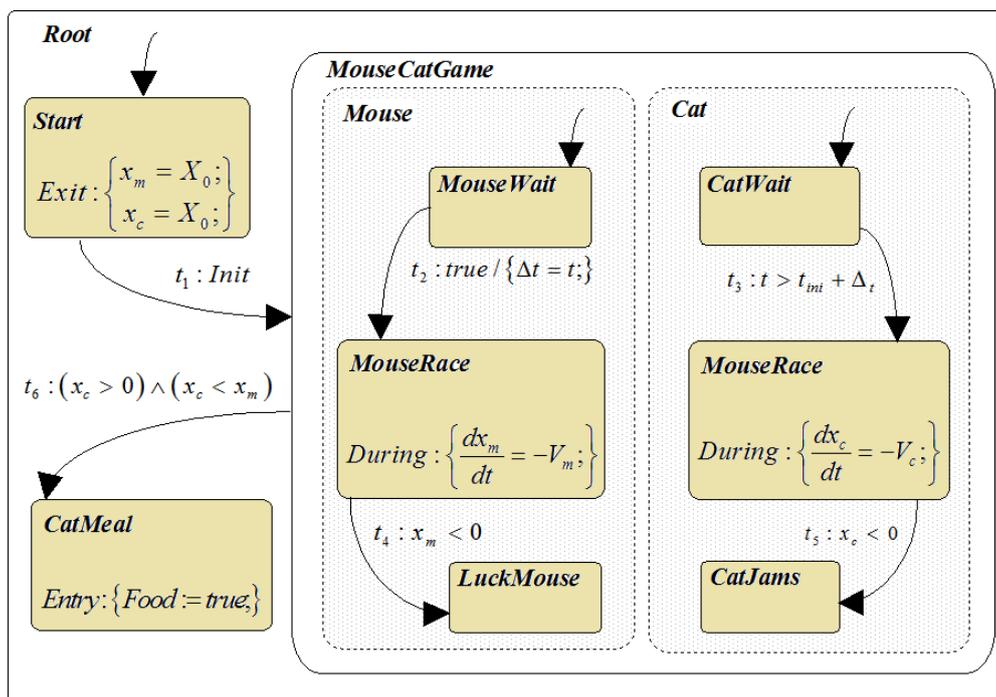


Abbildung 2.2.6: Hybride Statecharts

Hybride Petri-Netze

Hybride Petri-Netze (HPN) stellen eine Erweiterung der diskreten Petri-Netze um den kontinuierlichen Aspekt dar (Abbildung 2.2.7). Kontinuierliche Stellen sind anstatt der Marken mit reellen positiven Zahlen versehen ([All98]). Weitere Formen sind das Hybrid Dynamic Net ([Dra98]) oder das Hybrid Functional Petri Net ([Mat03]).

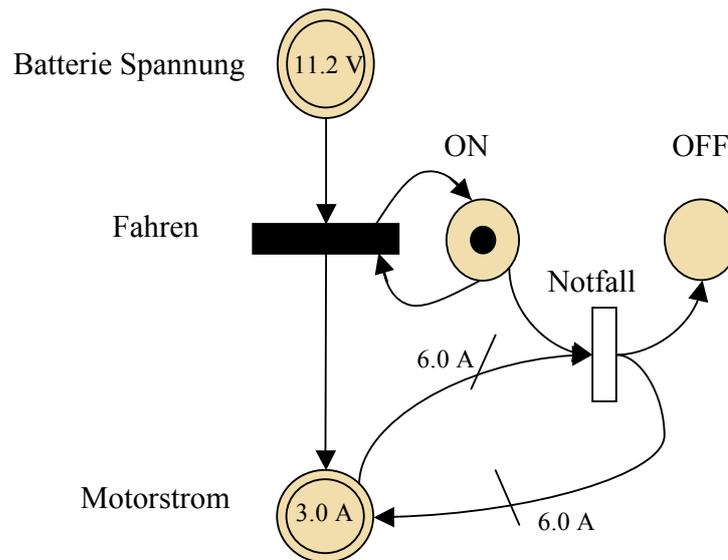


Abbildung 2.2.7: Hybride Petri-Netze

2.3 Überwachung und Fehlerdiagnose

Zur Entwicklung eines geeigneten Verfahrens zur Überwachung komplexer dynamischer Systeme sind folgende Schritte erforderlich: (1) Modellierung, (2) Beobachtung und (3) Fehleranalyse. Bei der Modellierung wird ein mathematisches Modell gebildet. Zur Fehlererkennung wird das durch dieses Modell berechnete Verhalten mit dem gemessenen Verhalten des realen Prozesses verglichen, um daraus Fehler und Ausfälle zu erkennen.

2.3.1 Aufgabe der Überwachung

Unter Überwachung wird ein Prozess verstanden, der nebenläufig mit einem System oder einer Komponente ausgeführt wird und diese beaufsichtigt, Daten aufnimmt und analysiert. Ziel der Überwachung ist es, auftretende Fehler in einem technischen Prozess zu erkennen und mögliche Ursachen zu ermitteln. Somit ist die Fehlerdetektion eine Teilaufgabe des Überwachungsprozesses. Basierend auf dem Verhaltensmodell und den gemessenen Werten des zu überwachenden Systems erfolgt eine Vorhersage des künftigen erwünschten Verhaltens. Tritt hierbei ein Widerspruch mit dem realen Verhalten auf, wird ein Fehler gemeldet. Ein Fehler ist die Abweichung des Verhaltens eines Systems von einem vorgegebenen (in der Spezifikation) erwünschten Verhalten. Dies bedeutet, dass ein spezifiziertes Verhalten nicht angeboten wird. Fehler haben früher oder später Störungen und Ausfälle zur Folge. Eine Überwachung soll dazu dienen, diese Störungen und Ausfälle möglichst zu vermeiden ([Ise04]-[Ise06]), indem Abweichungen im Systemverhalten frühzeitig erkannt und gemeldet werden.

Die Klassifizierungskriterien der Fehler sind sehr unterschiedlich und wurden in [Lap92], [Pet90] und [Sto96] zusammengefasst. Aus praktischer Sicht unterscheidet man zwischen internen und externen Fehlern ([Lun03]). Interne Fehler betreffen die Systemkomponenten selbst und werden in der Praxis häufig, in Abhängigkeit von ihrem Eingriffsort, in Akteurfehler, Systemfehler und Sensorfehler unterteilt. Externe Fehler betreffen Änderungen in der Umgebung bzw. in den Umweltbedingungen eines Systems, aufgrund dessen sich das System anders verhält als erwartet. Somit ist nicht das System fehlerhaft, sondern die Funktionsfähigkeit des Systems ist durch Einflüsse aus der Umgebung beeinträchtigt. In dieser Arbeit erfolgt die Klassifizierung der Fehler in interne sowie externe, und zwar unter Berücksichtigung der Spezifikation und der Umgebungsannahmen. Wird ein Fehler erkannt, d. h. ein spezifiziertes Verhalten wird nicht angeboten, wird das System mittels Rekonfiguration in einen sicheren Zustand gebracht.

Die Vorgehensweise bei einer Überwachung und Fehlerdiagnose lässt sich in einen dreistufigen Prozess einteilen (Abbildung 2.3.1):

- 1) *Echtzeit-Überwachung und Fehlererkennung*: Durch den kontinuierlichen Vergleich der gemessenen Daten mit den Referenzwerten werden Abweichungen vom erwünschten Systemverhalten und deren Zeitpunkten erkannt. Diese Differenz wird auch Residuum genannt. Bewegt sich das Residuum innerhalb eines vorher festgelegten, zulässigen Bereichs, so kann von einem bestimmungsgemäßen Betrieb ausgegangen werden. Überschreitet das Residuum jedoch einen Schwellwert, wird von einem Fehler ausgegangen, und ein Alarm wird ausgelöst. Der nächste Schritt umfasst die Alarmanalyse.
- 2) *Alarmanalyse*: Diese Phase umfasst eine Residuenauswertung und einen Entscheidungsprozess. Bei der Residuenauswertung wird die Abweichung in Bezug auf bestimmte Kriterien (Schwellwerte, Sicherheitsmarge) untersucht, durch die sich der Fehler lokalisieren bzw. die Fehlerart bestimmen lässt. Im Entscheidungsprozess geht es weiter mit einer Analyse der Beziehungen zwischen den Symptomen und den potenziellen physikalischen Ursachen der Fehler. Dabei werden Entscheidungen über die Fehlerursachen getroffen.
- 3) *Rekonfiguration / Sofortige Maßnahmen*: Diese Phase umfasst die Fehlerkorrektur, indem geeignete Gegenmaßnahmen eingeleitet werden. Eine Möglichkeit besteht darin, die Reglerparameter oder –struktur durch eine Rekonfiguration zu ändern. Eine weitere und in der Praxis häufig günstigere Alternative ist es, das System durch sofortige Maßnahmen (z.

B. System anhalten) in einen sicheren Zustand zu überführen. In dieser Arbeit wird es um diese Variante gehen.

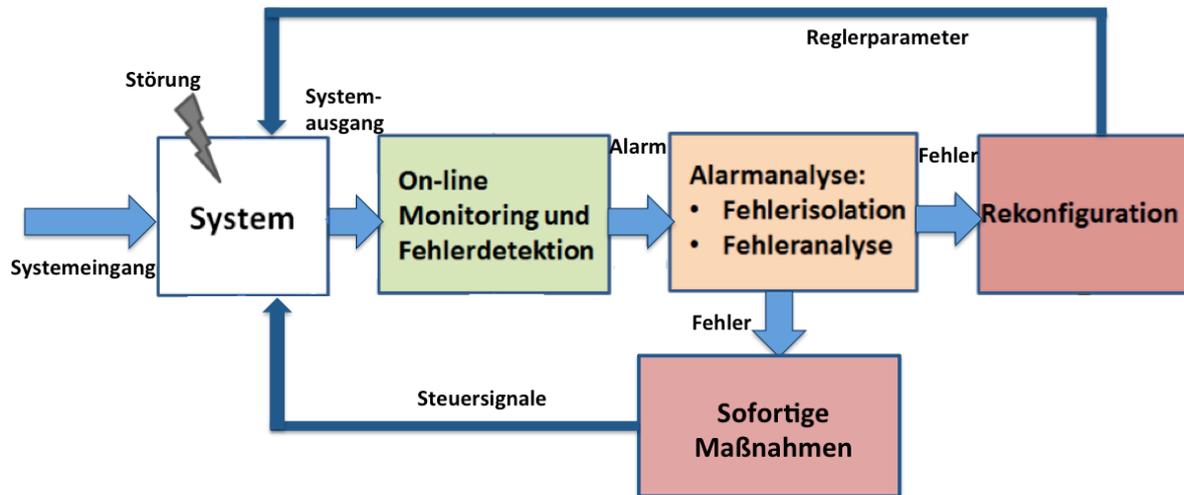


Abbildung 2.3.1: Überwachung und Fehlerdiagnose

2.3.2 Verfahren der Fehlerdiagnose, Defizite und Anforderungen

Es existieren zahlreiche Verfahren zur Fehlerdiagnose, die in der Literatur behandelt werden und hier vorgestellt werden:

- 1) *Die signalbasierten Verfahren* sind am weitesten verbreitet ([Fra94]). Diese erkennen Fehler direkt aus gemessenen Größen ([Lau99, Ise94]). Sobald die zu überwachenden Größen (z. B. Füllstand, Temperatur, Geschwindigkeit) die vorgegebenen Grenzwerte unter- bzw. überschreiten, wird ein Alarm ausgelöst. Allerdings ist diese Art der Überwachung in vielen Fällen nicht ausreichend. Deshalb ist es sinnvoller, Modelle, die das reale System nachbilden, zu verwenden.
 - 2) *Die modellbasierten Verfahren* haben den Vorteil, dass im Vergleich zu der Grenzwertüberwachung zum einen mehr Fehler entdeckt und zum anderen Fehler auch „vorhergesagt“ werden können. In der modellbasierten Überwachung und Diagnose gibt es wiederum verschiedene Ansätze: *die quantitativen und die qualitativen Lösungsansätze*.
- *Die qualitativen Modellierungsverfahren* stammen aus dem Bereich der Künstlichen Intelligenz, und sie beschäftigen sich mit der qualitativen Auswertung des Systemverhaltens, basierend auf qualitativen Angaben ([Ven03a]-[Ven03b]). Es handelt sich um das so genannte „Qualitative Schließen“. Die Grundidee dabei ist, Begriffe aus

der natürlichen menschlichen Sprache zu nutzen und sich dem menschlichen Schließen anzunähern. Hierfür werden die kontinuierlichen Systemgrößen entweder in definierte Wertebereiche bzw. Intervalle unterteilt oder durch Zugehörigkeitsfunktionen der Fuzzy-Logik als Fuzzy-Variablen dargestellt. Für das „Qualitative Schließen“ wird jedem diskreten Bereich ein Symbol oder ein linguistischer Begriff zugeordnet.

- *Die quantitativen Modellierungsverfahren* verwenden dynamische Modelle des zu überwachenden Systems ([Ven03a]-[Ven03b]). Diese mathematischen Modelle erlauben eine genaue Beschreibung des Systemverhaltens. Sie drücken die Abhängigkeiten der Eingangs- und Ausgangssignale, in Form von Differential- und Differenzgleichungen, aus. Anhand der gemessenen Signalwerte können auch interne Größen, wie Parameter und Zustände, geschätzt werden ([Pat00]). Man unterscheidet deswegen zwischen zwei Arten der Fehlerdiagnose (Abbildung 2.3.2):
 - a) *Dynamische Fehlerdiagnose ohne Schätzung*: Das Verhalten des realen Systems wird direkt mit dem Verhalten des Modells verglichen. Dabei bekommt das Modell dieselben Eingangsgrößen, die auch auf das reale System einwirken und liefert einen Satz idealer Ausgangsgrößen. Diese werden mit den real gemessenen Ausgangsgrößen verglichen, und aus der Differenz entsteht das Residuum.
 - b) *Dynamische Fehlerdiagnose mit Schätzung*: Die inneren Größen eines Systems, die nicht direkt messbar sind, können anhand gemessener Größen und durch die Verwendung einiger Algorithmen geschätzt werden. Diese internen Größen sind Zustandsgrößen und Parameter des realen Systems. Die Zustandsschätzung und Parameterschätzung sind wichtige Hilfsmittel für die Überwachung.
- *Zustandsschätzung*: Zur Schätzung der nicht messbaren Systemgrößen wird ein Zustandsbeobachter eingesetzt. In der Regelungstechnik hat man für die Ermittlung von Systemgrößen einfache Zustandsbeobachtungen ([Pet06]), den so genannten Luenberger-Beobachter ([Bou99]-[Ale01]) bzw. Kalman-Filter ([Kal60]), im Fall stochastischer Störungen eingesetzt. Für die Zustandsschätzung soll das mathematische Modell als Zustandsraumdarstellung vorliegen. Damit werden anhand der Eingangsgrößen die Ausgangsgrößen geschätzt. Die Differenz zwischen den geschätzten Ausgangsgrößen und den real messbaren Ausgangsgrößen wird in das Modell eingespeist, um die Zustandsgrößen zu schätzen. Durch den Vergleich mit den Soll-Zustandsgrößen wird das Residuum gebildet und für die Fehleranalyse

weiterverwendet ([Coc12]).

- *Parameterschätzung*: Bei der Parameterschätzung wird ebenfalls ein Beobachtermodell, parallel zum realen Prozess, eingesetzt. Aus der Differenz der Ausgangsgrößen des Modells und der realen Ausgangsgrößen erfolgt eine Schätzung der Modellparameter, indem diese so lange verändert werden, bis die Summe der quadratischen Abweichung minimal wird. Aus den geschätzten Modellparametern können dann nachfolgend die Systemkoeffizienten geschätzt werden. Das Residuum ist die Änderung der Systemkoeffizienten gegenüber den Normalwerten.

Im weiteren Verlauf werden in dieser Arbeit als quantitative Modelle nur mathematische Modelle mit Zustandsschätzung eingesetzt. Diese Schätzung wird in der Überwachung verwendet, um eine Vorhersage des Systemverhaltens zu ermöglichen.

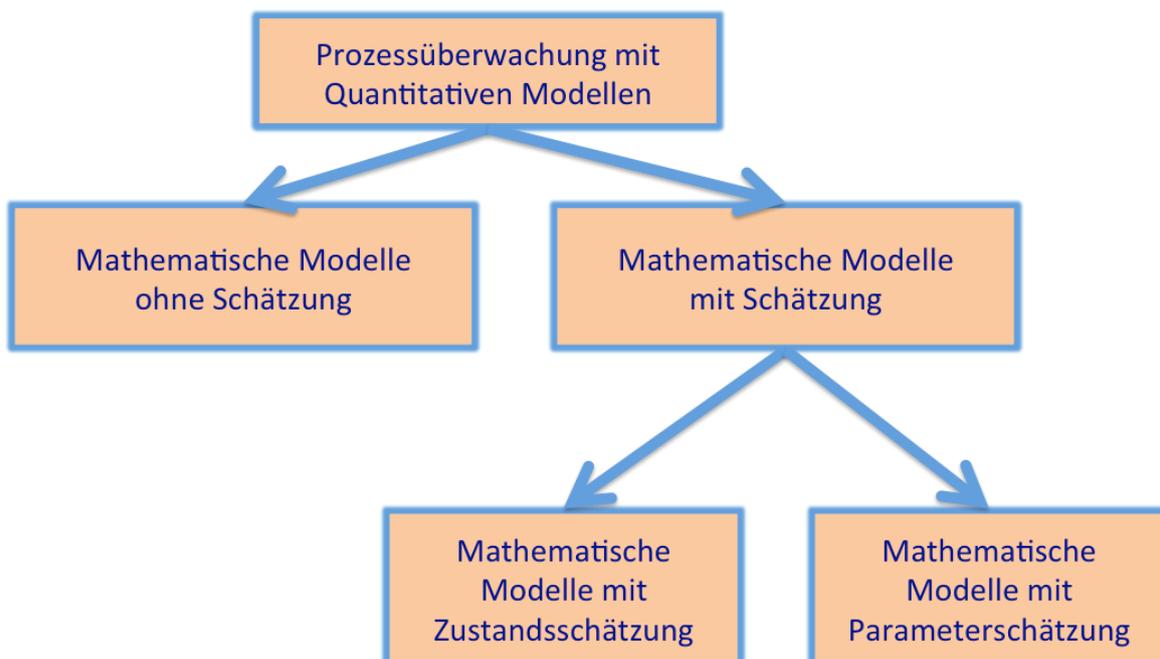


Abbildung 2.3.2: Quantitative Prozessüberwachung

Im Allgemeinen basieren Methoden der Überwachung mittels Zustandsschätzung auf zwei Schritten, nämlich: Die Residuumgenerierung und die Residuumevaluation [Abd06]. Bei der Residuumgenerierung wird der gemessene Wert mit dem, gemäß des mathematischen Modells, erwarteten Wert verglichen, um Abweichungen vom Idealzustand festzustellen. Im fehlerfreien Fall soll dieser Wert bei null liegen ([Sim03]). Die Aufgabe der Residuumevaluation ist es, Residuen zu analysieren, z. B. durch den Vergleich mit einer

ausgewählten Schwelle, und damit Fehler festzustellen, wenn diese Schwelle überschritten wird ([Bla03]). Allerdings könnte dieser Wert auch im fehlerfreien Fall, aufgrund von Rauschen in den Signalen und Modellfehlern von null abweichen [Ger98], was eine weitere Analyse dieser Abweichung erfordert. Auf diese Problematik wird im Verlauf dieser Arbeit eingegangen.

2.4 Defizite im Stand der Technik

In den letzten drei Jahrzehnten wurden im Bereich der Überwachung und Fehlerdiagnose viele Forschungen durchgeführt. Es wurden sowohl Verfahren für ereignisdiskrete Systeme als auch für kontinuierliche Systeme vorgeschlagen. Allerdings zeigen diese Ansätze Einschränkungen, wenn es um hybride Systeme geht. Arbeiten, die den hybriden Aspekt berücksichtigt haben, werden im Folgenden vorgestellt.

2.4.1 Multi-Model Filter

a) Multi-Model Kalman-Filter

In [Vee04] beruht die Schätzung auf einer Reihe von Kalman-Filtern, die jeweils einen bestimmten Betriebsmodus des Systems verfolgen. Der höchstwahrscheinlichste Zustand bestimmt den höchstwahrscheinlichsten Filter und damit den höchstwahrscheinlichsten Modus des Systems (z. B. das höchstwahrscheinlichste Zustand eines Fahrzeuges, wie Abbiegung, Beschleunigung usw.). Auch in [Rou98] wurde dieser Ansatz für einen mobilen Roboter verwendet: Modelle für Systemverhalten bei unterschiedlichen Arten von Fehlern wurden in den verschiedenen parallelen Kalman-Filtern eingebettet. Allerdings können die Ausfälle in beliebiger Kombination auftreten, weswegen die Zahl der Kalman-Filter exponentiell wächst. Das Multi-Model Kalman-Filter ist also wenig geeignet für die Schätzung hybrider Systeme, da man dabei viel mehr Filter als Modi verwendet.

b) Partikel-Filter für hybride Systeme

Diese Technik wird für die Überwachung von mobilen Robotern ([Dea02]-[Ver04]) in Echtzeit und beim Auftreten von Unsicherheiten verwendet. Partikel-Filter haben allgemein einen großen Vorteil für die Schätzung hybrider Systeme ([Blo04]), denn sie sind einfach zu implementieren, und sie verfügen über mehrere Partikel, die sich unabhängig entwickeln. Im Gegensatz dazu folgt eine Gauss-Verteilung (Kalman-Filter) nur einer Dynamik. Die einzigen Nachteile der Partikel-Filter sind:

- Es gibt keine Modellierung der Übergänge zwischen den verschiedenen Modi.

- Die Beobachtung ist nur numerisch. Es gibt keine Beobachtung des symbolischen Betriebsmodus oder des Ereignisempfangs.

Aus diesem Grund wurde diese Technik in weiteren Arbeiten mit diskreten Modellen gekoppelt.

2.4.2 Hybride bayesische Netze

Bei der Modellierung der hybriden Systeme mit bayesischen Netzen ([Ler02]) erfolgt die Berechnung des Systemzustands durch eine rekursive Schätzung, auf Grundlage der Regel von Bayes. Diese Methode zeigt jedoch dieselben Nachteile wie die dynamischen bayesischen Netze: mangelnde Verständlichkeit der Modelle und eine Begrenzung der Unsicherheiten. Einige Arbeiten ([Dou01]) koppeln die bayesischen Netze zur Modellierung des Systemverhaltens mit einem Partikel-Filter. Das dynamische bayesische Netz ermöglicht die Modellierung des Schätzungsprozesses: Es ermöglicht einerseits die Berechnung der Wahrscheinlichkeit des Systemzustands und andererseits, im Resampling-Schritt, berechnet es den Likelihood-Wert der Partikel. Dies erlaubt eine Beurteilung des Verhaltens des hybriden Systems. Die Kopplung zwischen bayesischen Netzen und Partikel-Filtern bringt mehr Informationen, im Vergleich zu einer reinen Partikel-Filterung. Jedoch ist diese Modellierung stochastisch (wegen der Notwendigkeit der Zuordnung von Wahrscheinlichkeiten in Zuständen und Transitionen), und die Darstellung ist schwer zu begreifen.

2.4.3 Diskrete Modelle und numerische Filter

- Probabilistische hybride Automaten

Die Arbeit von Hofbaur und Williams ([Hof02]-[Hof04]) basiert auf hybriden probabilistischen Automaten. Das Systemverhalten wurde mit einer Reihe von probabilistischen hybriden Automaten modelliert. Für die Schätzung des Systemzustands wurde ein Satz von Kalman-Filtern verwendet. Jedem Filter wurde eine mögliche Systemtrajektorie zugeordnet. Um eine wachsende Komplexität der Methode zu vermeiden, die durch die Verwendung eines Filters pro Trajektorie entsteht, wurde ein Auswahlverfahren durchgeführt: Nur die Filter der höchstwahrscheinlichen Trajektorien werden beibehalten, und die restlichen Filter werden entfernt. Diese Arbeit verwendet nur probabilistische Übergänge und erlaubt die Schätzung des diskreten Modus eines Systems aus der Beobachtung seines kontinuierlichen Zustands und nicht aus der Beobachtung des Modus.

- Hybride Automaten und Partikel-Filter

Koutsoukos et al. in [Kou03] verwenden hybride Automaten mit Partikel-Filtern zur Abschätzung des Modus eines hybriden Systems. Jedes Partikel trägt einen Zustandsvektor und einen Modus. Die Partikel-Filterung führt ein Resampling der besten Partikel durch. Dies wird verwendet, um festzustellen, welcher Modus des Systems der höchstwahrscheinlichste ist. Diese Methode behandelt nicht die Unsicherheit über die Modi des Systems, da nur der wahrscheinlichste Modus in Erwägung gezogen wird. Außerdem wird, wie in den oben genannten Arbeiten, die Beobachtung des Modus nicht behandelt. [Cha13] hat das hybride Diagnoseverfahren von [Bay08] um das probabilistische Weibull-Modell von [Rib11] erweitert, um ein hybrides Diagnoseverfahren in einem probabilistischen Framework zu entwickeln. Der Ansatz kombiniert hybride Automaten mit stochastischen. Allerdings war der größte Nachteil dieses Ansatzes, dass eine Explosion des diskreten Zustandsraums aufgetreten ist.

- Petri-Netze und Partikel-Filter

In der Literatur sind viele Petri-Netz-Modelle für die zentrale Systemüberwachung zu finden ([Sri93]-[Jul00]-[Ham09]), die aber Unsicherheiten nicht behandeln. Eine verteilte Überwachung mit Petri-Netzen findet man nur in [Zha01]. Allerdings handelt es sich hierbei um ein Zeit-Petri-Netz (Timed petri net), das nur das diskrete Verhalten eines Systems modelliert und die Schätzung des Systemzustands auf die zeitliche Entwicklung beschränkt. Nur wenige Arbeiten haben die Vorteile der Zustandsschätzung mit den Vorteilen der Petri-Netze kombiniert ([Yan03]-[Les07]). In diesen Arbeiten findet jedoch keine Berücksichtigung der Unsicherheiten, in beiden Teilen der Dynamik, statt.

2.5 Probabilistische Zustandsschätzung

Eine Möglichkeit, ein System zu überwachen ist, seine Zustände aus Beobachtungen, welche in Form von Sensormessungen vorliegen, zu schätzen. Meistens wird dieses Problem, im Fall einer kontinuierlichen Schätzung eines dynamischen Zustandes, als „Tracking“ Problem bezeichnet ([NaC04]). Bei der probabilistischen Zustandsschätzung werden die beteiligten Systemzustände als Zufallsgrößen betrachtet und die Systemprozesse durch probabilistische Modelle beschrieben ([Hal03]). In vielen Anwendungen werden die Messwerte sequentiell beobachtet, und die Schlussfolgerungen sollen zur Laufzeit gemacht werden. Aus diesem Grund sind sequentielle oder rekursive Verfahren am besten geeignet, da in diesen Verfahren das Wissen über den Systemzustand x_{t-1} zum Zeitpunkt $t-1$ die Grundlage für die Schätzung des aktuellen Zustands x_t zum Zeitpunkt t bildet. Sei $y'_t = \{y_1, \dots, y_t\} = y_{1:t}$ die Sequenz der stochastisch-unabhängigen Beobachtungen bis zum Zeitpunkt t und $X_t =$

$\{x_1, \dots, x_t\}$ die Sequenz der dazugehörigen Systemzustände. Die Entwicklung des Systemzustandes über die Zeit beschreibt die Systemdynamik. Die Wahrscheinlichkeitsdichtefunktion eines aktuellen Zustandes wird, basierend auf den vorherigen Zuständen, $X_{t-1} = x_{0:t-1}$ aufgestellt. Das zentrale Element dieser mathematischen Modellierung bildet das Bayes-Theorem, woraus sich die Formeln für die so genannte Rekursive Bayes-Schätzung ableiten lassen ([Hau05]). Das Prinzip dieser Theorie beruht auf einer rekursiven Darstellung der Wahrscheinlichkeitsverteilung des gesuchten Zustandes. Hierbei wird aus einer neuen Information (eine neue Sensormessung) die vorherige Schätzung aktualisiert, und zwar ohne die vorausgegangenen Beobachtungen erneut einzubeziehen. Mit Hilfe des Satzes von Bayes lässt sich die Posteriori-Verteilung $P(x_t | y)$ zu jeder Zeit t in folgende Komponenten aufspalten:

$$P(x_{0:t} | y_{1:t}) = \frac{P(y_{1:t} | x_{0:t}) P(x_{0:t})}{\int P(y_{1:t} | x_{0:t}) P(x_{0:t}) dx_{0:t}} \quad (2.5.1)$$

Unter der Markov-Annahme hängt die Wahrscheinlichkeitsdichtefunktion von x_t nur von seinem direkten Vorgänger x_{t-1} und nicht von der gesamten Historie $X_{t-1} = x_{0:t-1}$ ab, das heißt $P(x_t | x_{0:t-1}) = P(x_t | x_{t-1})$. Dadurch und mit Hilfe des Satzes von Bayes erhält man diese Rekursion:

$$\begin{aligned} P(x_{0:t+1} | y_{1:t+1}) &= \frac{P(y_{1:t+1} | x_{0:t+1}) P(x_{0:t+1})}{P(y_{1:t+1})} \\ &= \frac{P(y_{1:t}, y_{t+1} | x_{0:t+1}) P(x_{0:t+1})}{P(y_{1:t}, y_{t+1})} \\ &= \frac{P(y_{1:t} | x_{0:t+1}) P(y_{t+1} | y_{1:t}, x_{0:t+1}) P(x_{0:t+1})}{P(y_{t+1} | y_{1:t}) P(y_{1:t})} \\ &= \frac{P(y_{1:t} | x_{0:t+1}) P(y_{t+1} | y_{1:t}, x_{0:t+1}) P(x_{0:t+1})}{P(x_{0:t+1}) P(y_{t+1} | y_{1:t}) P(y_{1:t})} \\ &= \frac{P(x_{0:t+1} | y_{1:t}) P(y_{t+1} | y_{1:t}, x_{0:t+1})}{P(y_{t+1} | y_{1:t})} \\ &= \frac{P(x_{0:t+1} | y_{1:t}) P(y_{t+1} | x_{0:t+1})}{P(y_{t+1} | y_{1:t})} \\ &= \frac{P(x_{0:t} | y_{1:t}) P(x_{t+1} | x_{0:t}, y_{1:t}) P(y_{t+1} | x_{0:t+1})}{P(y_{t+1} | y_{1:t})} \end{aligned}$$

$$= P(x_{0:t}|y_{1:t}) \frac{P(x_{t+1}|x_t) P(y_{t+1}|x_{0:t+1})}{P(y_{t+1}|y_{1:t})} \quad (2.5.2)$$

Zur Implementierung dieser theoretischen Beschreibung werden spezielle Bayes-Filter für spezifische Problemklassen eingesetzt. Die bekanntesten Varianten sind der Kalman-Filter (KF), der Erweiterte Kalman-Filter (EKF) ([San08]) und der Partikel-Filter (PF). In dieser Arbeit werden die rekursive Bayes-Schätzung und der darauf aufbauende Partikel-Filter implementiert. Diese einführenden Beschreibungen und Erläuterungen basieren zum Teil auf [Dou01] und [Aru02]. Abbildung 2.5.1 zeigt das Prinzip des kontinuierlichen Tracking und die Idee, die Verteilung $P(x|y)$ mit Hilfe der Kontrolldaten u_t und der Sensordaten y_t zu berechnen:

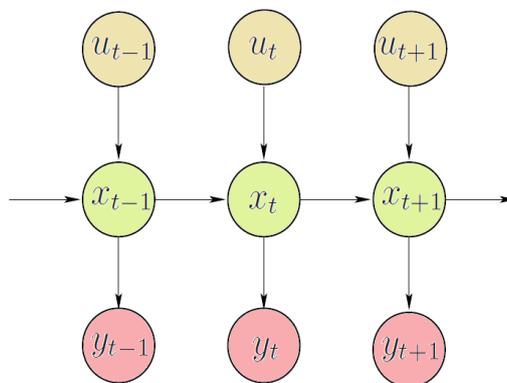


Abbildung 2.5.1: Das Tracking-Problem

Der Systemzustand x_t wird zum Zeitpunkt t über eine Abbildungsfunktion h und anhand der aktuellen Beobachtung y_t und der Historie $x_{1..t-1}$ bzw. $y_{1..t-1}$ geschätzt. Allerdings kann unter der vereinfachenden Markov-Annahme die Schätzung von x_t allein anhand von y_t und x_{t-1} ermittelt werden. Es handelt sich bei dieser probabilistischen Zustandsschätzung um einen zweistufigen Prozess (Abbildung 2.5.2):

- a) Ein Vorhersageschritt (prediction): Aus dem vergangenen Zustand und dem Vorwissen um die Systemdynamik wird eine a-priori Wahrscheinlichkeitsverteilung $P^-(x)$ für den aktuellen Zustand x_t gebildet.

$$P(x_t|y_{1:t-1}) = \int P(x_t|x_{t-1}, y_{1:t-1}) P(x_{t-1}|y_{1:t-1}) dx_{t-1} \quad (2.5.3)$$

- b) Aktualisierungsschritt (update): Hier kommt die aktuelle Beobachtung in Betrachtung, und aus der a-priori wird eine a-posteriori Wahrscheinlichkeitsverteilung $P^+(x)$. Der Erwartungswert dieser Wahrscheinlichkeitsverteilung kann als momentane Schätzung ausgegeben werden.

$$P(x_t | y_{1:t}) = \frac{P(y_t | x_t) P(x_t | y_{1:t-1})}{\int P(y_t | x_t) P(x_t | y_{1:t-1}) dx_t} \quad (2.5.4)$$



Abbildung 2.5.2: Prozesse der Zustandsschätzung

2.5.1 Sequentielles Monte-Carlo-Sampling

Die SMC-Methode basiert auf dem Prinzip der Zufallsstichprobe und liefert eine approximative Schätzung der a-posteriori Wahrscheinlichkeitsverteilung, auch bei nicht-normalverteilten und nicht-linearen Beobachtungen. Eine ausführliche Beschreibung der SMC-Methoden geben [Dou01]. Beim Partikel-Filter stellt jedes Partikel eine diskrete Hypothese des Systemzustands dar, woraus eine beste Schätzung gewonnen wird. Jede Hypothese beschreibt einen potenziellen Zustand, in dem das System sich befinden könnte und wird durch eine Menge von N gewichteten Stichproben mit ihren zugehörigen Gewichten abgebildet:

$$\Pi = \{(x^i, w^i) \mid i = 1, \dots, N\}. \quad (2.5.5)$$

Ein Partikel π^i besitzt ein Gewicht w^i . Alle Gewichte zusammen ergeben die Summe 1. Die Menge der Partikeln wird durch die folgende Verteilung beschrieben:

$$P(x) = \sum_{i=1}^N w^i \delta(x^i) \quad (2.5.6)$$

$\delta(\cdot)$ ist die Dirac Delta Funktion. Zur Bestimmung der Partikelmenge zu einem Zeitpunkt t werden aus den N alten Partikeln π_{t-1}^i neue N Partikeln π_t^i gemäß ihrer Gewichte w^i zufällig gezogen. Sie propagieren mit Hilfe des dynamischen Modells $P(x_t | x_{t-1})$. Diese Stichprobe bildet eine Näherung für $P(x_t | y_{t-1})$, die mit einer aktuellen Beobachtung y_t durch neue Gewichte $w_t^i \propto P(y_t | x_t = \pi_t^i)$ berechnet wird. $P(y_t | x_t = \pi_t^i)$ beschreibt das

Beobachtungsmodell. Die Schätzung des aktuellen Zustands ergibt sich aus dem Erwartungswert der Partikelmenge:

$$\hat{x}_t = E(x_t | y_t) = \sum_{i=1..N} w_t^i \pi_t^i \quad (2.5.7)$$

Die Schritte des Schätzungsprozesses sind folgende:

- Die Erzeugung der Partikel geschieht durch zufälliges Ziehen von Stichproben.
- Basierend auf einer Messung aus dem System (Sensoren), werden die Partikel, entsprechend ihrer Plausibilität, bezüglich der Sensordaten gewichtet.

Sequential Importance Sampling SIS

Der Sequential Importance Sampling Algorithmus ist entwickelt worden, um die Entwicklung der Partikel sukzessiv zu ermitteln ([Fun04]). Es berechnet demzufolge die Verteilung $P(x_t | x_{t-1})$ rekursiv, wie in der Abbildung 2.5.3 demonstriert wird. Mit $\pi(x_t | x_{0:t}, y_{0:k})$ wird die so genannte Importance *Sampling-Verteilung* bezeichnet.

Seien die Partikel $\{x_{0:t} : i = 1, \dots, N\}$ unabhängig, dann werden diese, gemäß der Sampling-Verteilung $\pi(x_t | x_{0:t}, y_{0:k})$ verteilt. Für die normalisierten Importance-Gewichte erhält man für die Rekursion:

$$\hat{w}_t^i \propto \hat{w}_t^i \frac{P(y_t | x_t^i) P(x_t^i | x_{t-1}^i)}{\pi(x_t^i | x_{0:t-1}^i, y_{1:t})} \quad (2.5.8)$$

Der Pseudo-Code des SIS Algorithmus ist in Abbildung 2.5.4 dargestellt. In der Praxis erweist sich allerdings die Anwendung des SIS Algorithmus als nicht optimal, da im Laufe der Zeit die „besten“ Partikel das Gewicht von 1 annehmen, während die anderen Partikel gegen null konvergieren. Oft kommt es in der Praxis vor, dass nur noch ein Partikel ein Importance-Gewicht ungleich null hat. Die anderen Partikel werden mit der Zeit mehr und mehr gestreut. Dies führt dazu, dass die Posteriori-Verteilung nicht hinreichend angenähert werden kann. Damit verfolgt man Partikel, welche fast keine Information über den Zustand liefern.

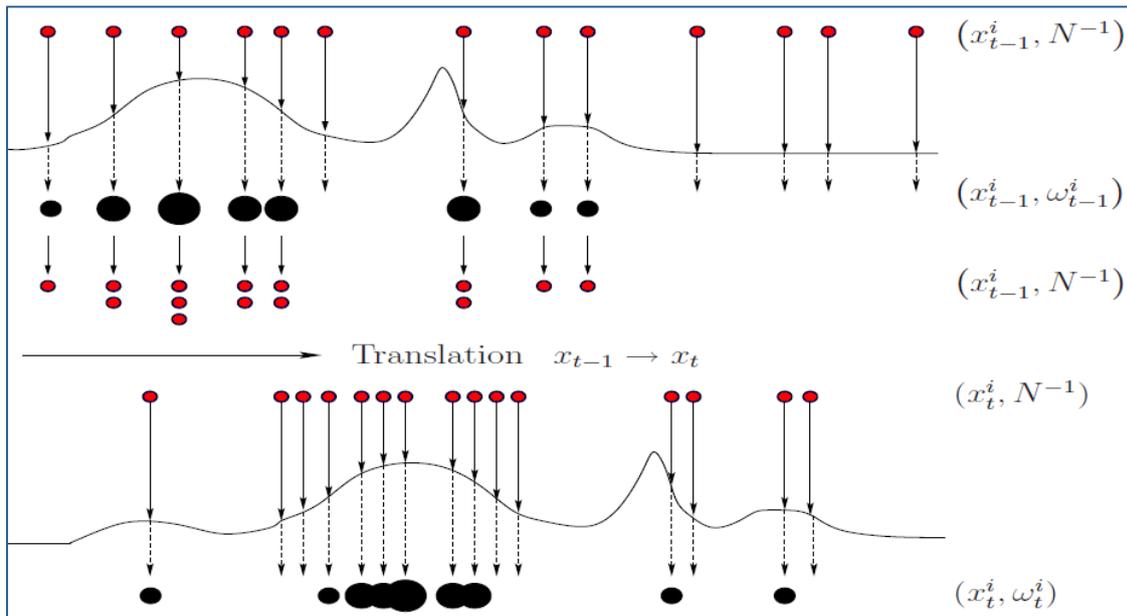


Abbildung 2.5.3: Sequential Importance Sampling Prozess [Fun04]

Abbildung 2.5.5 zeigt das Tracking-Beispiel eines mobilen Fahrzeugs bzw. Roboters und wie nach einer gewissen Zeitspanne nur noch wenige Partikel die wahre Position widerspiegeln können. Eine naive Lösung des Problems wäre die Anzahl der Partikel zu erhöhen, um deren Degeneration zu vermindern. Dieses Vorgehen ist allerdings wegen des hohen Rechenaufwands ([Hau05]) nicht immer praktikabel. Die Lösung dieses Problems ist der SIR „Sampling Importance Resampling“-Algorithmus im Partikel-Filter.

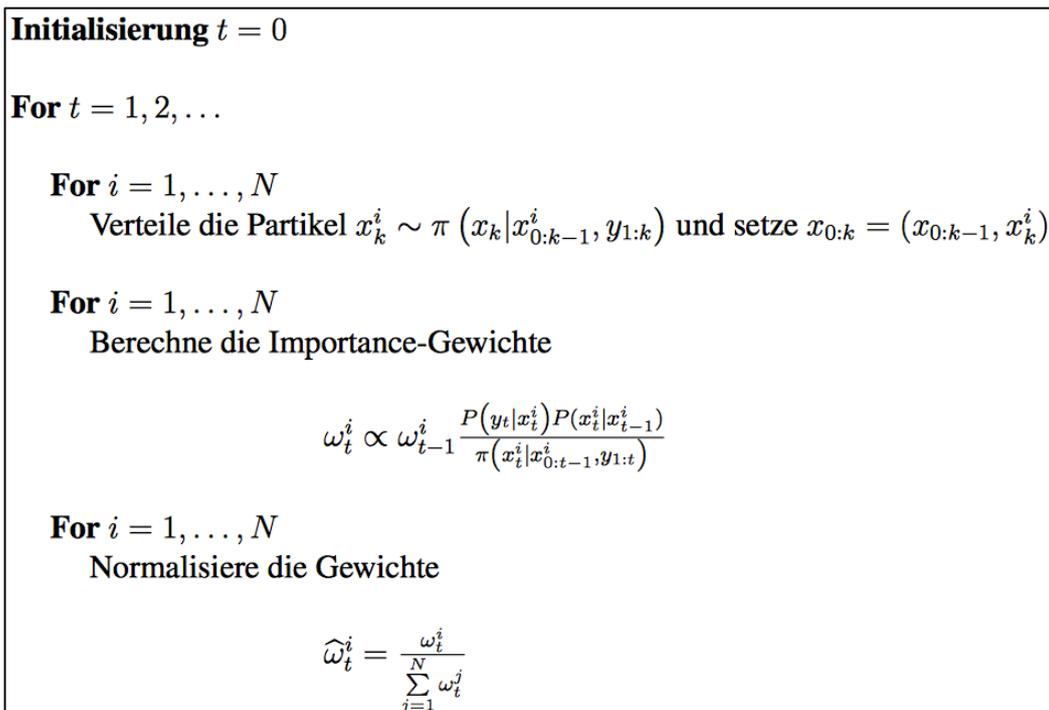


Abbildung 2.5.4: Algorithmus für Sequential Importance Sampling (SIS)

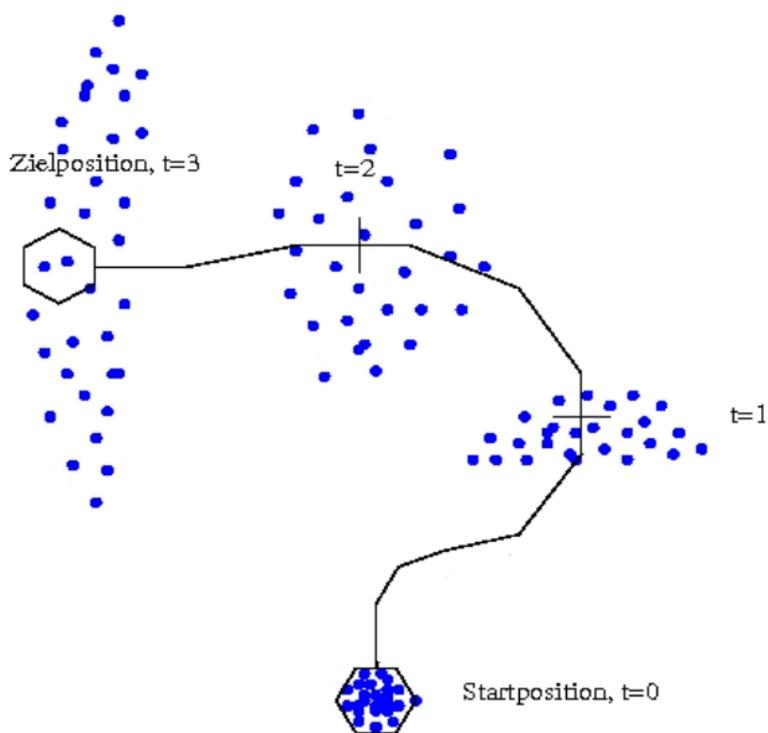


Abbildung 2.5.5: Streuung der Partikel über die Zeit

2.5.2 Partikel-Filter

Der Partikel-Filter stellt eine nicht-parametrische Implementierung des Bayes-Filters dar. Die geschätzte Verteilung wird durch die Menge der Partikel und nicht durch eine parametrische Funktion, wie beim Kalman-Filter, dargestellt. Somit ist die Zustandsschätzung multimodal und nicht unimodal (nur ein Wert). Dieses wahrscheinlichkeitstheoretische Schätzverfahren basiert auf dem Prinzip der Sequentiellen Monte-Carlo-Methode (SMC), die im vorherigen Abschnitt vorgestellt wurde.

Beim Partikel-Filter wird ein zusätzlicher Schritt durchgeführt, das so genannte „Resampling“ ([Jer06]), welches nur die besten Partikel aussucht und gleichmäßig gewichtet.

Sampling Importance Resampling SIR

Der SIR Algorithmus (Abbildung 2.5.6) unterscheidet sich vom SIS Algorithmus bei der Aktualisierung der Partikel. Der SIR Algorithmus basiert auf der so genannten Bootstrap Technik. Die Schlüsselidee dieser Technik ist, diejenigen Partikel mit geringem Gewicht \hat{w}_t^i zu löschen (Abbildung 2.5.7) und Partikel mit hohem Gewicht zu klonen ([Che03]). Dies bedeutet, dass das gewichtete Maß der Verteilung

$$P(x_{0:t}|y_{1:t}) = \sum_{i=1}^N \hat{w}_t^i \delta(x_{0:t}^i) \quad (2.5.9)$$

zu einem ungewichteten Maß wird:

$$P(x_{0:t}|y_{1:t}) = \sum_{i=1}^N N_t^i \delta(x_{0:t}^i) \quad (2.5.10)$$

N_t^i ist die Anzahl der überlebenden Partikel x_t^i falls $N_t^i > 0$. Bei $N_t^i = 0$ stirbt das Partikel. Die überlebenden Partikel werden bezüglich $P(x_{0:t}|y_{1:t})$ verteilt. Die Gewichte \hat{w}_{t-1}^i zum Zeitpunkt $t - 1$ tauchen beim Berechnen der neuen Gewichte zum Zeitpunkt t nicht auf, da die Gewichte aller Partikel beim Resampling-Schritt auf denselben Wert gesetzt werden.

Initialisierung: $t = 0$.

Für $i = 1, \dots, N$

Verteile die Partikel x_0^i bezüglich $P(x_0)$ und setze $t = 1$.

Importance-Sampling:

Für $i = 1, \dots, N$

Verteile das Partikel \hat{x}_t^i bezüglich $P(x_t|x_{t-1}^i)$
und füge es der Menge $\hat{x}_t^i = \hat{x}_{t-1}^i \cup \hat{x}_t^i$ hinzu.

Für $i = 1, \dots, N$

Berechne die Importance-Gewichte

$$\hat{w}_t^i = P(y_t|\hat{x}_t^i).$$

Normalisiere die Importance-Gewichte.

Resampling:

Ziehe mit Zurücklegen aus der Menge $(\hat{x}_t^i; i = 1, \dots, N)$ N Partikel proportional zu den Importance-Gewichten
und füge sie in die Menge $(x_t^i; i = 1, \dots, N)$ ein.

Setze $t = t + 1$, und führe wieder ein Importance Sampling durch.

Abbildung 2.5.6: Algorithmus für Sampling Importance Resampling (SIR)

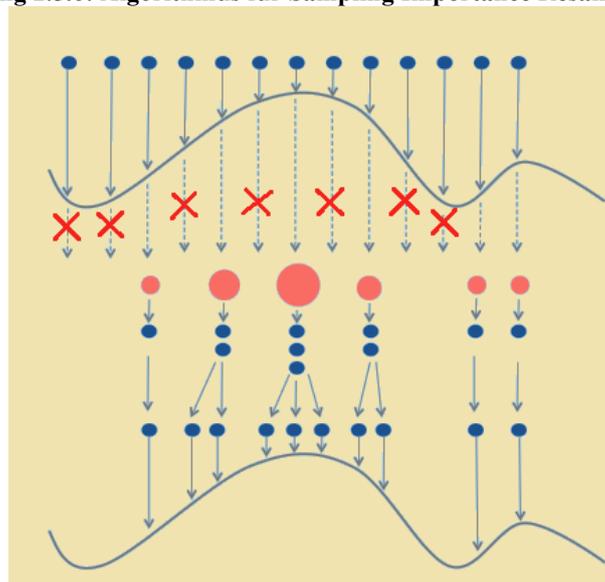


Abbildung 2.5.7: Das Prinzip des SIR

2.6 Petri-Netze für die Modellierung

Petri-Netze stellen ein verbreitetes Basiskonzept zur Modellierung, Analyse und Simulation von dynamischen Prozessen, auch mit nebenläufigen Vorgängen, dar. Sie wurden im Jahr 1962 von Prof. Dr. Carl Adam Petri entwickelt [Pet62], und sie basieren auf einer graphischen intuitiven Darstellung mit einfachen Grundbausteinen. Seitdem wurden sie in verschiedenen Bereichen, u. a. in der Wirtschaftslehre, Robotik, im Ingenieurwesen, in der Elektronik und Informatik eingesetzt. Als allgemeines mathematisches Modell unterstützen die Petri-Netze die Modellierung von Prozessen durch die Beschreibung der Beziehungen zwischen Bedingungen und Ereignissen, der Modellierung von Systemstrukturen und der mathematischen Analyse von dynamischen Merkmalen des Systems ([Ros91]). Petri-Netze werden auch zur Analyse für verteilte bzw. parallele Programme eingesetzt. Es gibt viele verschiedene Petri-Netz-Typen ([Har81]-[Rei91]-[Jen91]-[Hum89]), zum Beispiel Stellen-Transitionen-Netze, Bedingungs-Ereignis-Netze, Prädikat-Transitionen-Netze, Farbige Petri-Netze, Algebraische Petri-Netze), Signal-Netzsysteme und Zeit-Petri-Netze. Im Folgenden werden zunächst die klassischen Petri-Netze und ihre Merkmale beschrieben. Anschließend werden die in dieser Arbeit erbrachten Erweiterungen für die Modellierung hybrider verteilter Systeme vorgestellt.

2.6.1 Grundlegende Elemente von Petri-Netzen

Ein diskretes Petri-Netz ist ein gerichteter Graph, bestehend aus zwei fundamentalen Elementen, so genannten Stellen P (Kreise) und Transitionen T (Balken oder Rechtecke), die durch Kanten (Pfeile) miteinander verbunden sind (Abbildung 2.6.1). Die Stellen modellieren die Zustände eines Systems oder seiner Komponenten. Die Transitionen modellieren die Zustandsübergänge, also die Bedingungen für den Übergang von einer Stelle zu einer anderen. Die Stellen und Transitionen sind durch die Kanten miteinander verbunden und legen damit die Struktur des Netzes und die Abhängigkeiten fest. Ein weiterer Baustein des Modells sind Marken (auch Token genannt), die bildlich durch schwarze Punkte repräsentiert werden. Marken werden durch eine Markierungsfunktion den Stellen zugeordnet und repräsentieren den veränderlichen Zustand des Systems. Dieses Markierungskonzept, bestehend aus den Stellen des Petri-Netzes und den Marken, beschreibt die dynamischen Vorgänge in einem Prozess bzw. System als Zustände und Zustandsänderungen.

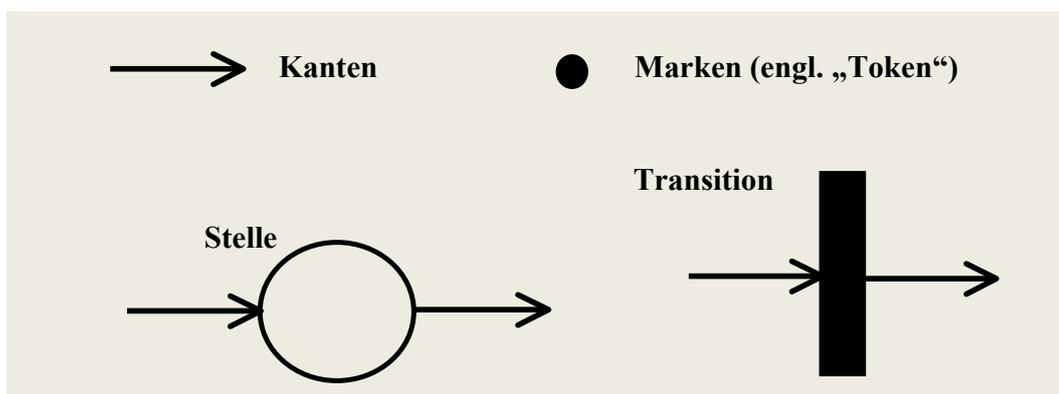


Abbildung 2.6.1: Grundlegende Elemente eines Petri-Netzes

Definition 2.6.1: Ein unmarkiertes gewöhnliches Petri-Netz (Abbildung 2.6.2) ist als gerichteter bipartiter Graph $Q = \langle P, T, F \rangle$ zu definieren, mit:

- $P = \{P_1, P_2, \dots, P_n\}$: ist eine nicht leere, endliche Menge von Stellen
- $T = \{T_1, T_2, \dots, T_m\}$: ist eine nicht leere, endliche Menge von Transitionen mit
- F : sind die Flussrelationen, die kausale oder zeitliche Vor-, Nachbedingungen von Übergängen aus T repräsentieren, $F \subseteq P \times T \cup T \times P$.

Beispiel:

- $P = \{P_1, P_2, P_3, P_4, P_5\}$
- $T = \{T_1, T_2, T_3, T_4, T_5\}$
- $F = \{(P_1, T_1), (T_1, P_2), (P_2, T_2), (T_2, P_3), (P_3, T_3), (T_3, P_4), (P_4, T_4), (T_4, P_1), (T_1, P_5), (P_5, T_4), (P_2, T_5), (T_5, P_4)\}$

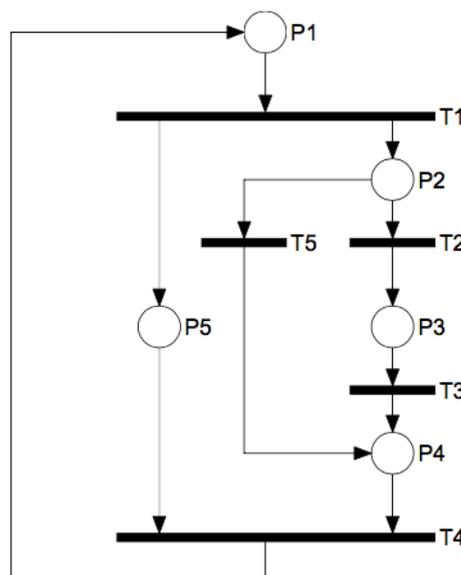


Abbildung 2.6.2: Unmarkiertes Petri-Netz

Der Hauptvorteil der Petri-Netze liegt in der Beschreibungsmöglichkeit von konkurrierenden, parallelen, synchronen sowie sequentiellen Prozessen. Petri-Netze können stets zerlegt werden, so dass sie aus den Elementen in Tabelle 2.6.1 aufgebaut werden können.

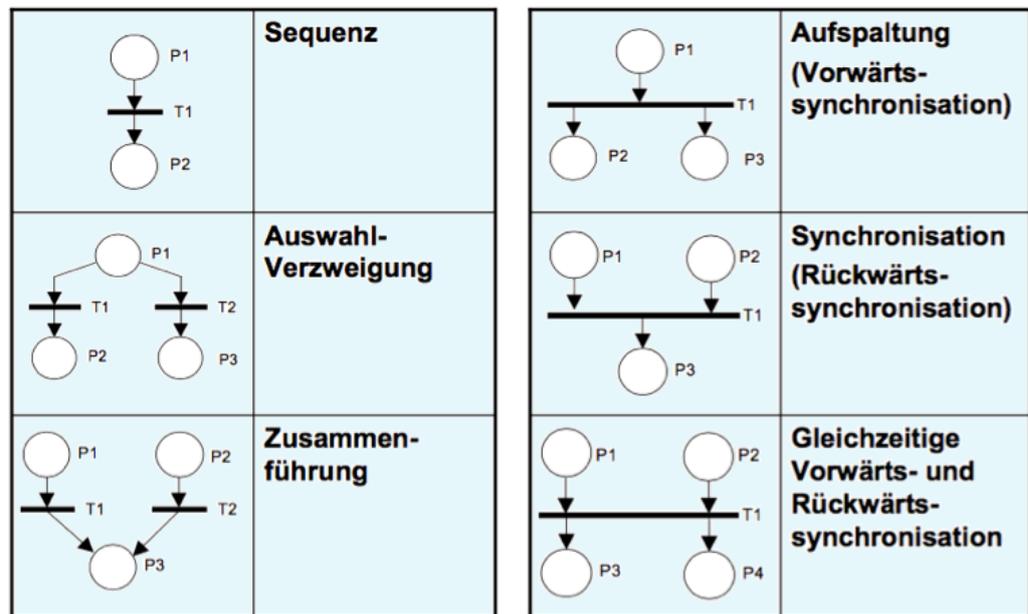


Tabelle 2.6.1: Aufbau von Petri-Netzen

Definition 2.6.2: Jede Stelle p , die eine Kante zu einer Transition t_i besitzt, wird Eingangsstelle oder Pre-Stelle genannt. Die Menge dieser Eingangsstellen bildet den *Vorbereich* ${}^{\circ}t_i$ der Transition $t_i \in T$:

$${}^{\circ}t_i = \{p \in P : (p, t_i) \in F\}. \quad (2.6.1)$$

Definition 2.6.3: Jede Stelle p , mit einer eingehenden Kante von einer Transition t_j , wird Ausgangsstelle oder Post-Stelle genannt. Die Menge dieser Ausgangsstellen bildet den *Nachbereich* t_i° der Transition $t_i \in T$:

$$t_i^{\circ} = \{p \in P : (p, t_i) \in F\}. \quad (2.6.2)$$

Definition 2.6.4: Die Inzidenzmatrix des Netzes ergibt sich aus: $\forall p \in P$ und $\forall t \in T$ $C(p, t) = Post(p, t) - Pre(p, t)$, mit:

- $Pre: P \times T \rightarrow \{0, 1\}$ definiert die nicht leere Menge V aller von Stellen p zu Transitionen t gerichteten Kanten, $Pre(p, t) \in V$.
- $Post: T \times P \rightarrow \{0, 1\}$ definiert die nicht leere Menge N der von Transitionen t zu Stellen p gerichteten Kanten, $Post(p, t) \in N$.

wobei $Pre(P_i \times T_j) = 1$ bedeutet, dass es eine Kante von $P_i \rightarrow T_j$ gibt und $Post(P_i \times T_j) = 1$, dass eine Kante von $T_j \rightarrow P_i$ existiert, mit $i \in \{1, 2, \dots, n\}$ und $j \in \{1, 2, \dots, m\}$. Die Kanten verbinden Plätze mit Transitionen oder Transitionen mit Plätzen, jedoch nie zwei Plätze oder Transitionen miteinander.

2.6.2 Modellierung dynamischer Verhalten durch Petri-Netze

Petri-Netze können auch zur Modellierung dynamischer Verhalten verwendet werden. Hierfür sollen Marken (Tokens) auf den Plätzen hinzugefügt und Schaltregeln zum Transport der Marken durch die Transitionen definiert werden. Abhängig vom gewählten Markierungskonzept entstehen verschiedene Arten von Petri-Netzen. Man unterscheidet z. B. zwischen diskreten schwarzen Petri-Netzen, die eine oder mehrere (schwarze), nicht unterscheidbare Marken auf einem Platz enthalten. Andere Arten von Petri-Netzen besitzen mehrere individuelle, unterscheidbare (farbige) Marken auf einem Platz. Diese werden gefärbte Petri-Netze (engl., coloured Petri nets) genannt. Bei Prädikat-Transitionen-Netzen PrT-Nets wird die Markierung durch prädikate logische Ausdrücke dargestellt. Eine interessante Variante für die Modellierung von hybriden Systemen sind die Hybriden Petri-Netze, die innerhalb der diskreten Stellen reelle Zahlen verwenden.

Definition 2.6.5: Ein markiertes Petri-Netz ist ein Quadrupel $PN = \langle P, T, F, m_0 \rangle$, bestehend aus dem unmarkierten Petri-Netz Q und einer binären Anfangsmarkierung $m_0: P \rightarrow \{0, 1\}^{|P|}$. Strukturell gleiche Netze mit unterschiedlicher Anfangsmarkierung ergeben unterschiedliches Verhalten, deswegen ist die Anfangsmarkierung ein wichtiger Teil der Netzdefinition.

Definition 2.6.6: Eine Markierung m_k des Netzes zum Zeitpunkt k ist eine Funktion $m: P \rightarrow \mathbb{N}_0$, die jeder Stelle eine natürliche Zahl zuordnet. Das Ergebnis ist ein Vektor, der den Systemzustand beschreibt. Die Markierung der Stellen $P_i = (p_1, \dots, p_n)$ ist durch den Vektor $m_k(P_i) = (m_1, \dots, m_n)$ beschrieben, welcher die Anzahl der Marken auf den einzelnen Stellen zum Zeitpunkt k angibt.

Beispiel: Abbildung 2.6.3 zeigt die Markierung eines Petri-Netzes zu einem bestimmten Zeitpunkt. Für die Stellen 1 bis 5 ergibt sich die Markierung (1, 2, 1, 0, 1). Die Stellen p_1 , p_2 und p_3 stellen den Vorbereich ${}^\circ t$ von t dar. Die Stellen p_4 und p_5 bilden den Nachbereich.

Definition 2.6.7: Eine Transition t_j ist konzessioniert (engl. enabled), wenn jede Stelle in ihrem Vorbereich ${}^\circ t_j$ mindestens eine Marke besitzt, d. h.:

$$\forall p_i \in {}^\circ t_j \quad m_k(p_i) \geq Pre(p_i, t_j) \quad (2.6.3)$$

Eine konzessionierte Transition kann (muss aber nicht) feuern (schalten). Mehrere gleichzeitig konzessionierte Transitionen feuern einzeln nacheinander aber niemals gleichzeitig.

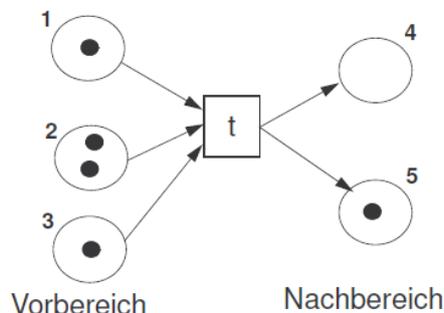


Abbildung 2.6.3: Markierung, Vorbereich und Nachbereich eines Petri-Netzes

Definition 2.6.8: Nur eine konzessionierte Transition t darf schalten, dabei entsteht aus der Markierung m eine Folgemarkierung m' . Es gilt:

$$\begin{cases} m'(p) = m(p) - 1 & \text{für } p \in {}^{\circ}t \\ m'(p') = m(p') + 1 & \text{für } p' \in t^{\circ} \\ m'(p) = m(p) & \text{sonst} \end{cases} \quad (2.6.4)$$

Die Markierung m in einem Platz des Petri-Netzes beschreibt eine Situation. Durch das Schalten einer Transition können mehrere Situationen in verschiedene neue Situationen überführt werden. Die Menge dieser aktiven Situationen ergibt den Zustand des Petri-Netzes.

Beispiel:

Sei m_0 eine Initialmarkierung eines Petri-Netzes PN und T eine feuerbare Transition (Abbildung 2.6.4). Das Feuere der Transition T_1 führt von einer Markierung m_0 (links) zu einer Markierung m_1 (rechts): $m_0 \xrightarrow{T_1} m_1$. Dabei wird aus den einzelnen Stellen im Vorbereich eine Marke entfernt und allen einzelnen Stellen im Nachbereich eine Marke hinzugefügt. Somit ergibt sich aus der Initialmarkierung eine neue Markierung.

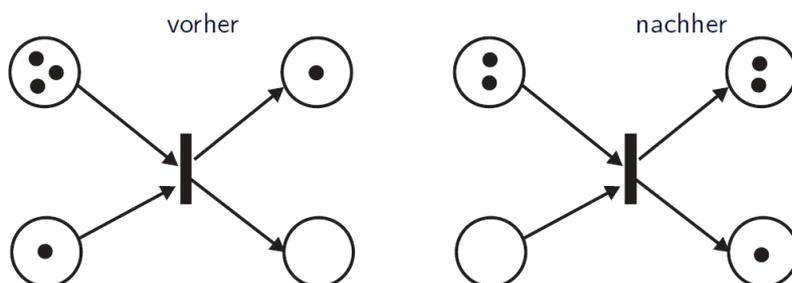


Abbildung 2.6.4: Markierung nach dem Schalten einer Transition

Definition 2.6.9: Erreichbarkeit: Das Schalten einer konzessionierten Transition t unter einer Markierung m in einem PN erzeugt eine neue Markierung m' . Man sagt: m' ist direkt erreichbar von m oder m' ist eine Folgemarkierung von m . Wenn eine Transition t' schaltet und somit aus m' eine Folgemarkierung m'' erzeugt, dann ist m'' von m erreichbar über die Schaltsequenz $\langle t, t' \rangle$.

Definition 2.6.10: Erreichbarkeitsgraph (reachability graph): Dies ist ein Graph E mit den erreichbaren Markierungen als Knoten. Die Kanten des Graphen werden mit der entsprechenden Transition beschriftet, die Knoten mit den Markierungen (Abbildung 2.6.5).

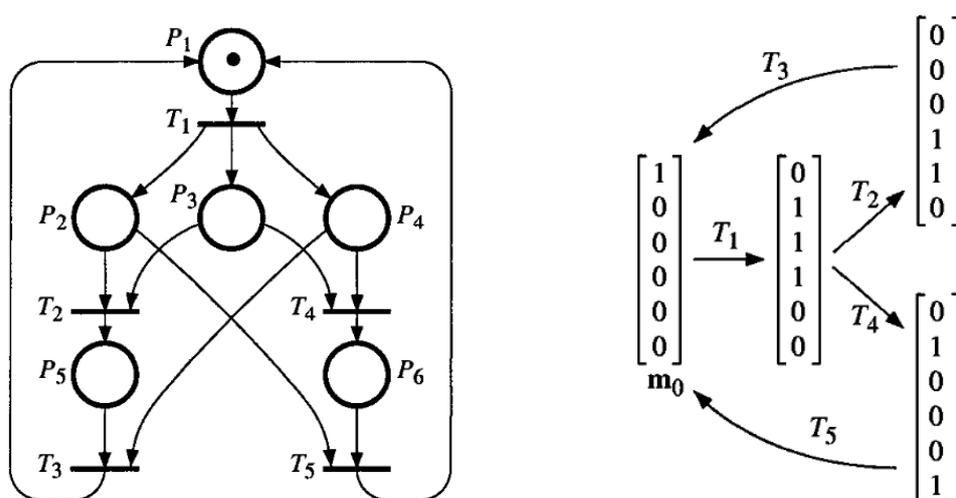


Abbildung 2.6.5: Erreichbarkeitsgraph eines Petri-Netzes ([Dav05])

2.6.3 Diskrete, kontinuierliche und hybride Petri-Netze

Die Haupteigenschaft eines kontinuierlichen Petri-Netzes ist das Markierungskonzept des Netzes ([Dav87]). Im Gegensatz zu den im vorherigen Abschnitt vorgestellten diskreten Petri-Netzen, die eine Ganzzahl an schwarzen Marken besitzen, wird die Markierung von kontinuierlichen Petri-Netzen als reelle Zahl dargestellt. Dadurch lassen sich kontinuierliche Systeme modellieren ([Dav05]). Abbildung 2.6.6 zeigt den Unterschied zwischen einem diskreten und einem kontinuierlichen Petri-Netz.

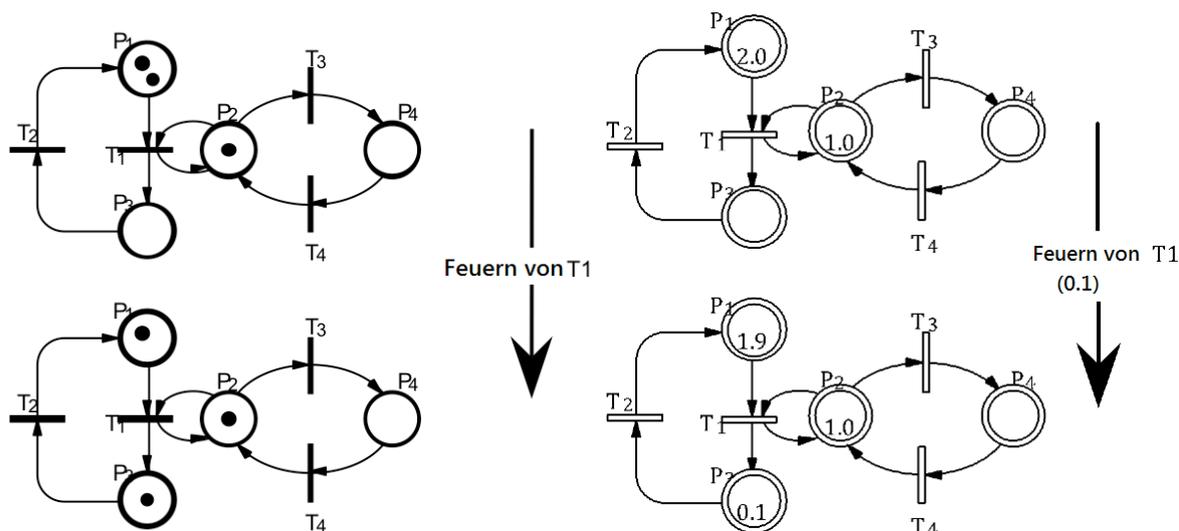


Abbildung 2.6.6: Diskrete (links) und kontinuierliche (rechts) Petri-Netze ([Dav05])

Hybride Petri-Netze wurden in [LeB91] vorgestellt. Diese dienen der Darstellung und Analyse hybrider Systeme und sind eine Kombination von diskreten und kontinuierlichen Petri-Netzen (Abbildung 2.6.7).

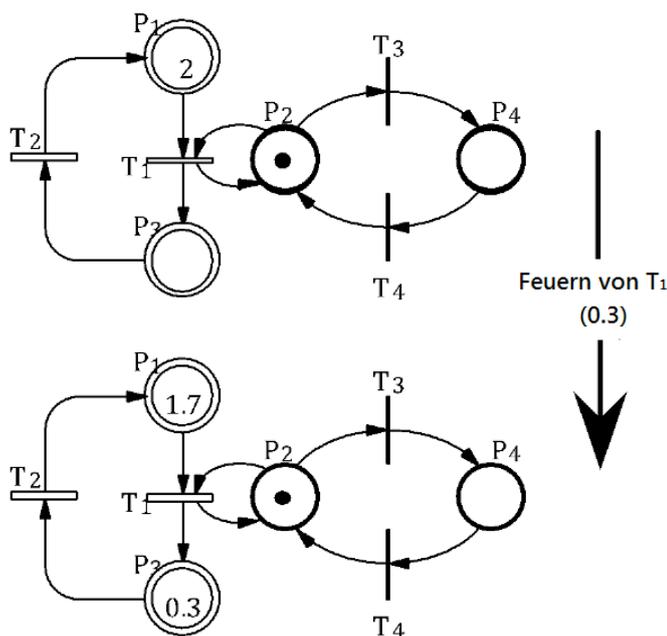


Abbildung 2.6.7: Hybride Petri-Netze ([Dav05])

Definition 2.6.11: Ein hybrides Petri-Netz ist ein Tupel $T = \langle P, T, Pre, Post, m_0, h \rangle$ ([Dav01]-[Dav05]):

- $P = \{P_1, P_2, \dots, P_n\}$ ist eine endliche, nicht leere Menge von Stellen,
- $T = \{T_1, T_2, \dots, T_m\}$ ist eine endliche, nicht leere Menge von Transitionen,
- mit $P \cap T = \emptyset$,

- $h: P \cap T \rightarrow \{D,C\}$ ist die so genannte hybride Funktion für jeden Platz, wobei D für „diskret“ und C für „kontinuierlich“ steht,
- $Pre: P \times T \rightarrow R^+$ oder N ,
- $Post: P \times T \rightarrow R^+$ oder N ,
- $m_0: P \times R^+$ oder N

Die Eigenschaften von hybriden Petri-Netzen verbinden die Charakteristiken von diskreten und kontinuierlichen Petri-Netzen. Diese Komponenten können einander jedoch beeinflussen. Die Beziehung der Elemente zueinander lässt sich in drei Kategorien unterteilen [Dav05]:

- Einfluss der diskreten Komponente auf die kontinuierliche Komponente,
- Einfluss der kontinuierlichen Komponente auf die diskrete Komponente,
- Konvertierung von kontinuierlicher in diskrete Markierung und umgekehrt.

Abbildung 2.6.8 zeigt den Einfluss des diskreten Petri-Netzes auf das kontinuierliche. Es handelt sich um das Modell eines Fahrzeugs. Solange die Stelle „Ein“ aktiv ist, also durch eine schwarze Marke belegt ist, kann die Transition „Fahren“ gefeuert werden. Dementsprechend fließt Strom in den Motor und das Fahrzeug kann fahren.

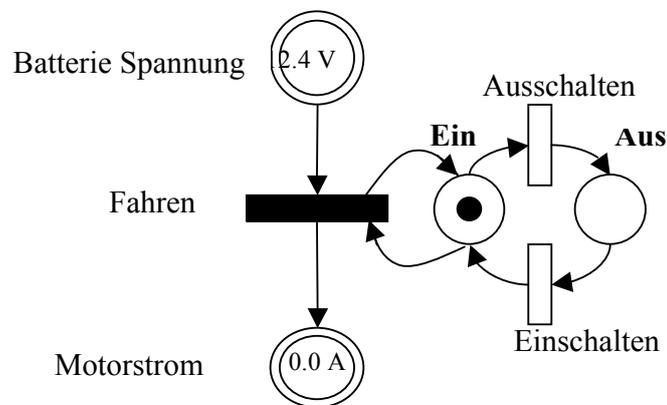


Abbildung 2.6.8: Einfluss der diskreten Komponente auf die kontinuierliche Komponente

Wird die Transition „Ausschalten“ aktiviert, so wandert die Markierung in die Stelle „Aus“. Demzufolge fließt kein Strom zum Elektromotor, und der Roboter hält an. Analog zu diesem Fall kann eine diskrete Komponente von einer kontinuierlichen abhängen. Dies wird durch Abbildung 2.6.9 veranschaulicht. Das Überschreiten der Motorspannung (6.0 A) führt zum Feuern der Transition „Not-Stop“. Dadurch wandert die Marke von der Stelle „Ein“ in die Stelle „Aus“. Demzufolge feuert die Transition „Fahren“ nicht mehr, und der Roboter hält an.

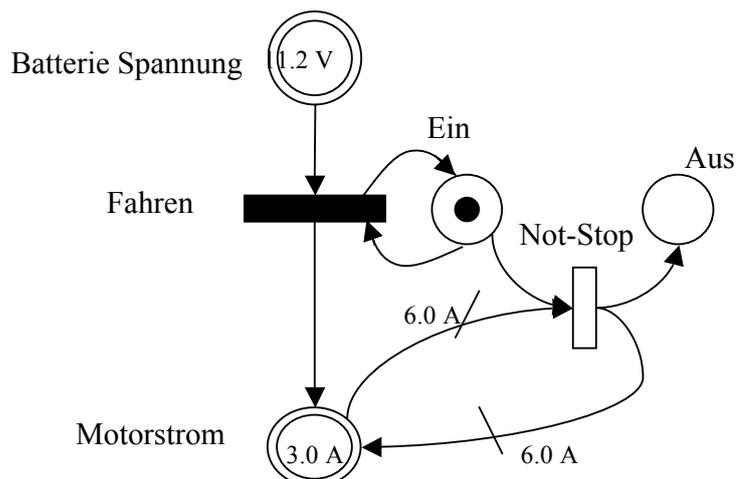


Abbildung 2.6.9: Einfluss der kontinuierlichen Komponente auf die kontinuierliche Komponente

In der dritten Variante beeinflussen sich beide Teile des hybriden Petri-Netzes gegenseitig. Sowohl diskrete Markierungen können in kontinuierliche Markierungen konvertiert werden als auch kontinuierliche in diskrete. In Abbildung 2.6.10 (Fall (a)) wird ein kontinuierlicher Zustand in einen diskreten umgewandelt. Ist die Batterie Spannung ≥ 12 V, so wird die Transition „Laden“ gefeuert, und die Stelle „Spannung OK“ wird mit einer schwarzen Marke belegt. Im Fall (b) feuert die Transition „Laden“, so wird eine Marke mit dem Wert 12 V an die Stelle „Batterie Spannung“ addiert.

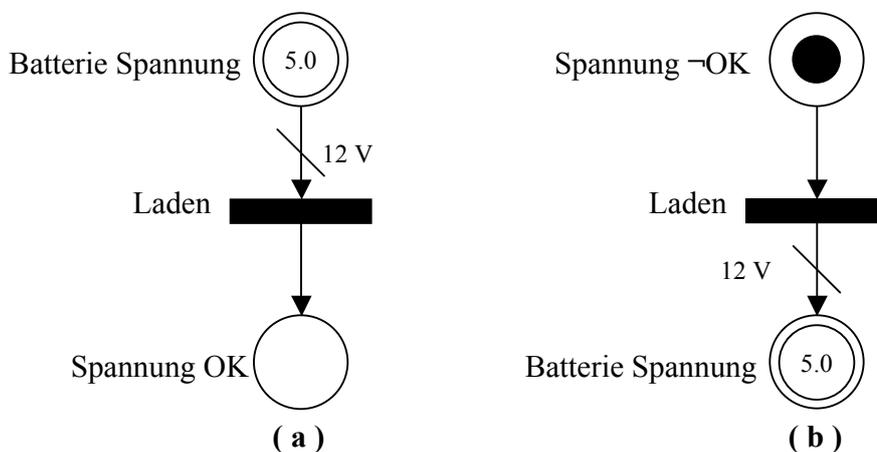


Abbildung 2.6.10: Konvertierung von kontinuierlicher in diskrete Markierung und umgekehrt

2.6.4 Possibilistische Petri-Netze

Partikel-Petri-Netze PPN wurden in [Les05] vorgestellt und dienen der hybriden Modellierung und Zustandsschätzung in einem Petri-Netzmodell, basierend auf einer possibilistischen und zum Teil einer probabilistischen Interpretation. PPN besitzen zwei Arten von Stellen bzw. Transitionen, analog zu Hybriden Petri-Netzen (HPN). Kontinuierliche

Stellen werden allerdings als numerische Stellen bezeichnet und diskrete Stellen als symbolische Stellen.

Definition 2.6.12: Ein Partikel-Petri-Netz PPN nach Lesire ([Les05]) ist definiert als: $T = \langle P, T, A, D, C, E, M_0 \rangle$. Es gilt:

- P ist die Menge der Stellen und es gilt $P = P_N \cup P_S$ sowie $P_N \cap P_S = \emptyset$, wobei $P^N \subset P$ die Menge aller numerischen Stellen und $P^S \subset P$ die Menge aller symbolischen Stellen ist.
- T ist die Menge aller Transitionen $T = T_N \cup T_S$ sowie $T_N \cap T_S = \emptyset$. T_N und T_S repräsentieren die numerische bzw. die symbolische Transitionen.
- A ist die Menge aller Kanten und es gilt $A \subset P \times T \cup T \times P$.
- D ist eine Menge von Differentialgleichungen, die den numerischen Stellen zugeordnet sind. Eine Differentialgleichung $D(p)$ repräsentiert die kontinuierliche Entwicklung der Markierung einer numerischen Stelle $p \in P_N$.
- C ist die Menge der Bedingungen, die den numerischen Transitionen zugeordnet sind. Die Bedingung $C(t)$ ist eine boolesche Funktion, die der numerischen Markierungen boolesche Werte zuordnet.
- E ist die Menge der Aktionen, assoziiert mit den symbolischen Transitionen. Der Effekt $E(t)$ ist eine Zuordnung von Parametern der symbolischen Token.
- M_0 ist die Anfangsmarkierung des Petrinetzes.

Ein Partikel $\pi_k^{(i)}$ ist ein numerischer Token im Zeitpunkt k . Jeder Partikel beschreibt einen möglichen numerischen Wert des Zustandsvektors des Systems. Eine Konfiguration $\delta_k^{(i)}$ ist ein symbolischer Token im Zeitpunkt k . Jede Konfiguration beschreibt einen Betriebsmodus des Systems. Die Hauptunterschiede zwischen HPN und PPN liegen in der Markierungs- und in den Feuerregeln [Les05].

Definition 2.6.13: Eine symbolische Transition ist nur dann aktiviert, wenn jeder seiner Eingänge markiert ist. Es gilt:

$$t \in T_S \text{ ist aktiviert} \Leftrightarrow \begin{cases} \forall p \in \bullet t \cap P_N, \exists \pi \in M(p) \\ \forall p \in \bullet t \cap P_S, \exists \delta \in M(p) \end{cases} \quad (2.6.5)$$

wobei $\bullet t$ die Menge aller Eingänge der Transition t ist. Analog dazu ist $t \bullet$ die Menge aller Ausgänge einer Transition t . Die symbolischen Transitionen feuern nach den folgenden Schaltregeln:

$$\forall p \in \bullet t \cap P_N, \forall \pi \in M(p), \forall p' \in t \bullet \cap P_N, M(p') := M(p) \cup \{\pi\} \quad (2.6.6)$$

$$\forall p \in \bullet t \cap P_S, \forall \delta \in M(p), \forall p' \in t \bullet \cap P_S, M(p') := M(p) \cup \{E(t)(\delta)\} \quad (2.6.7)$$

wobei $E(t)(\delta)$ die durch den Effekt $E(t)$ modifizierte Konfiguration δ ist. Analog zur symbolischen Transition gelten für numerische Transitionen die Regeln der Gleichung 2.6.8. Eine numerische Transition ist nur dann aktiviert, wenn jeder ihrer Eingänge durch einen Partikel, der die Bedingungen der Transition erfüllt, markiert ist.

$$t \in T_N \text{ ist aktiviert} \Leftrightarrow \left\{ \begin{array}{l} \forall p \in \bullet t \cap P_n, \exists \pi \in M(p) / C(t)(\pi) \\ \forall p \in \bullet t \cap P_s, \exists \delta \in M(p) \end{array} \right\} \quad (2.6.8)$$

wobei $C(t)(\pi)$ den booleschen Wert „Wahr“ annimmt, falls der Partikel π die Bedingung $C(t)$ erfüllt. Die Schätzung des zukünftigen Systemzustandes erfolgt in zwei Schritten [Les07]:

- Der Schritt der „isomarking evolution“ berechnet die zukünftigen Werte der Partikel nach der Differentialgleichung $F(k, p, \pi_{k|k}, v_k(p))$. In einem Korrekturschritt werden die geschätzten Werte mit einer Beobachtung z aus dem System verglichen und entsprechend gewichtet:

$$w(\pi_{k+1|k}^i) = \frac{P(z_{k+1} | \pi_{k+1|k}^i)}{\sum_{j=0}^N P(z_{k+1} | \pi_{k+1|k}^j)} \quad (2.6.9)$$

wobei z_{k+1} die Beobachtung zum Zeitpunkt $k + 1$ ist und $w(\pi)$ das Gewicht des Partikels π .

- Der Schritt der „isoparticle evolution“ berechnet alle zukünftigen möglichen symbolischen Zustände. Dieser Schritt beruht auf einer possibilistischen Interpretation, dem so genannten „Ranking“:

$$(z_{k+1} \rightarrow \delta_{k+1|k}^i) \wedge \neg(z_{k+1} \rightarrow \delta_{k+1|k}^j) \Rightarrow \delta_{k+1|k+1}^i \prec_{\wedge} \delta_{k+1|k+1}^j \quad (2.6.10)$$

Die Notation $\delta^i <_{\wedge} \delta^j$ besagt, dass eine Konfiguration δ^i bevorzugt oder mindestens äquivalent zu δ^j sein soll. Weiterhin bedeutet $z \rightarrow \delta$, dass die Konfiguration δ der Beobachtung z entspricht. Der Zustand mit den höchstgewichteten Partikeln $\{\pi_{K+1IK}^i, w(\pi_{K+1IK}^i)\}$ beschreibt den höchstwahrscheinlichsten Systemzustand.

Die PPN ([Les05]) mit ihren probabilistischen-possibilistischen Interpretationen werden für die Überwachung von Situationen verwendet. Eine Anwendung fand bei der Planung von Flugzeugprozessen statt (Aufstieg, Beschleunigung, Abbremsen, Landung). Abbildung 2.6.11 zeigt einen Modellausschnitt zu der Phase „Landung“.

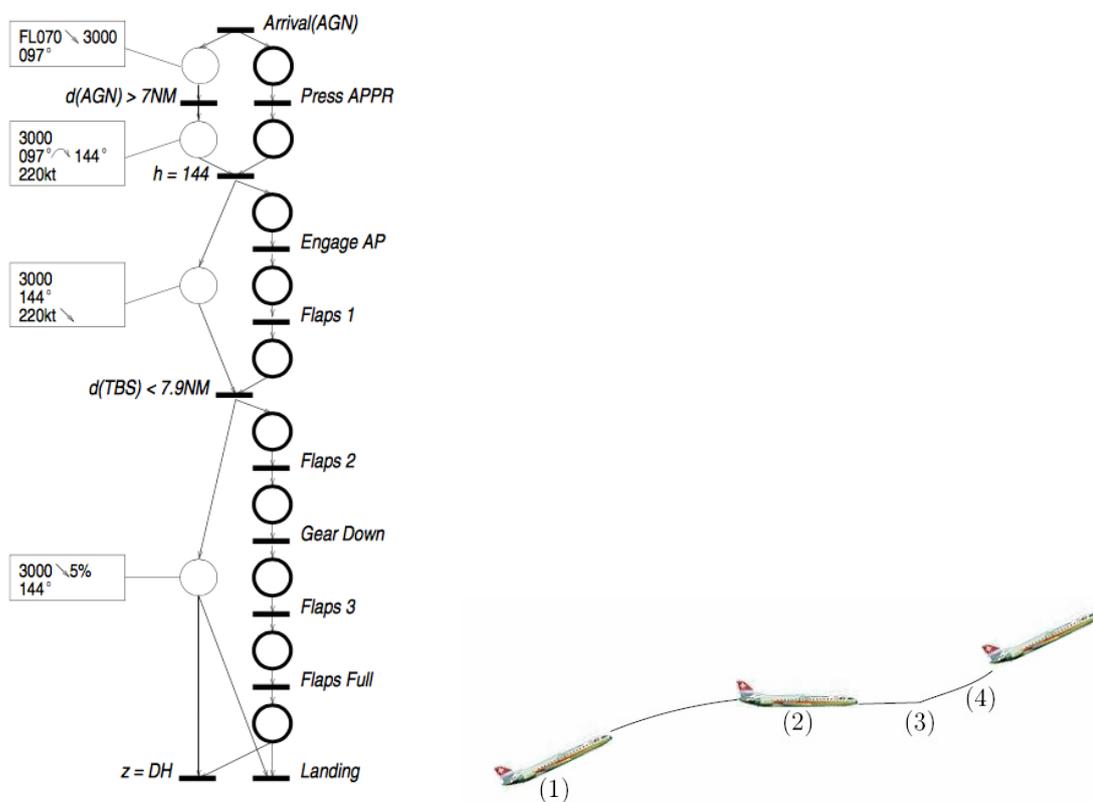


Abbildung 2.6.11: Überwachung der Prozessdurchführung an Flugzeugen ([Les05])

In den PPN finden beide Teile der Schätzung, die probabilistische und die possibilistische Schätzung, unabhängig voneinander statt. Daher soll an dieser Stelle ein neuer Ansatz entwickelt werden, der die wechselseitige Beeinflussung des kontinuierlichen und diskreten Teils eines Systems berücksichtigt.

Die Grundlagen des in dieser Arbeit entwickelten Überwachungsverfahrens stammen aus den klassischen Definitionen der PPN. Allerdings umfasst die Erweiterung auf die Modifizierten Partikel-Petri-Netze MPPN folgende Punkte: eine einheitliche probabilistische hybride

Schätzung, eine hybride Modellierung, die die Wechselwirkungen zwischen der kontinuierlichen und diskreten Dynamik berücksichtigt, sowie neue Feuerbarkeitsregeln.

Der Vorteil graphischer Modelle, wie die Petri-Netze, ist die anschaulichere Darstellung, allerdings ist die Nachbildung von Prozessen mit großen Modellen sehr aufwendig. Dieser Nachteil wird im Rahmen dieser Arbeit beseitigt, indem die graphischen Modelle automatisch in textuelle und mathematische Modelle umgeschrieben werden. Weitere methodische Beiträge, die durch die MPPN in dieser Arbeit erreicht werden, sind die Einführung neuer Konzepte, wie die Online Generierung der Überwachungsmodelle, die Beschreibung der Überwachungskomponente als Teil der Systemspezifikation und die Einbettung der Modelle in Echtzeit-Anwendungen, durch die Entwicklung eines echtzeitfähigen MPPN-Simulators.

2.6.5 Vergleich von Petri-Netzen mit endlichen Automaten

Endliche Automaten sind, wie die Petri-Netze, zustandsbasierte Modelle. Ein endlicher Automat wird durch Zustände (Kreise) und Übergänge (Pfeile) beschrieben. Die Zustandsübergänge hängen dabei von den Eingaben ab, und die Zustände selbst können Ausgaben erzeugen.

Definition 2.6.14: Ein endlicher Automat (engl., finite automata (FA), auch finite state machines (FSM)) wird als $EA = (E, A, Z, z_0, F, G)$ dargestellt, mit:

- E ist die endliche Menge der möglichen Eingaben,
- A ist eine endliche Menge der möglichen Ausgaben,
- Z ist die Menge der möglichen Zustände und z_0 ist der Anfangszustand.
- $F: Z \times E \rightarrow Z$ ist die Zustandsübergangsfunktion,
- $G: Z \times E \rightarrow A$ ist die Ausgabefunktion.

Wenn der Eingang eine Schaltbedingung erfüllt und somit ein bestimmtes Ereignis im Automaten auslöst, findet ein Übergang in einen neuen Zustand statt. Dieser Übergang wird durch die Zustandsübergangsfunktion F festgelegt. Für das Beispiel in Abbildung 2.6.12 gilt:

- $Z = \{0, 1, 2\}$ mit $z_0 = \{0\}$
- $E = \{a, b, c, d\}$
- $A = \{0, 1, k\}$

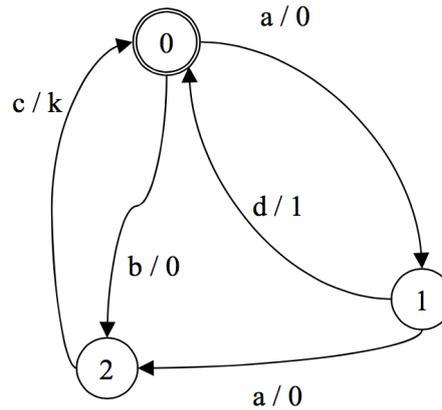


Abbildung 2.6.12: Endlicher Automat

Es gibt eine Vielzahl von Erweiterungen für die Automaten: Zeitautomaten (mit Übergangszeiten), stochastische Automaten (bei einer bestimmten Eingabe sind verschiedene Übergänge möglich), hybride Automaten (die Zustände werden durch kontinuierliche Zustandsgleichungen beschrieben, und Übergänge hängen von Werten der kontinuierlichen Zustandsgrößen ab). Die Automaten erlauben die Modellierung von sequentiellen Abläufen, Verzweigungen und Zusammenführungen. Allerdings, während bei Automaten nur Zustände als Knoten abgebildet werden, können Petri-Netze auch die Übergänge zwischen den Zuständen als explizite Knoten darstellen. Zusätzlich zu dieser wichtigen Eigenschaft erlauben die Petri-Netze, neben der Darstellung sequentieller Abläufe, die Beschreibung nebenläufiger Prozesse. Sie modellieren somit verteilte Systeme und spezifizieren sie mathematisch. Diese Modellierungsmächtigkeit der Petri-Netze macht sie für den Einsatz in verschiedensten Gebieten sehr geeignet.

3 Einheitlicher Ansatz zur hybriden Modellierung und Überwachung

Im Rahmen dieser Arbeit wird der verhaltensbasierte Ansatz für die Modellierung und Überwachung hybrider Systeme verfolgt. Dies erfordert eine Auseinandersetzung mit dem hybriden Formalismus bei der Verhaltensbeschreibung, den Fragestellungen der Topologie und des Abstraktionsgrades der Modelle. Aus diesem Grund wird zunächst in Abschnitt 3.1 eine formale Definition für dynamische hybride Systeme beschrieben. Diese Definition dient als Grundlage für den entwickelten Überwachungsansatz. In Abschnitt 3.2 wird der Mechanismus zur Fehlerdetektion, als Teilaufgabe des Überwachungsprozesses, erläutert. Die Architektur des Ansatzes wird in Abschnitt 3.3 präsentiert und in Abschnitt 3.4, in Zusammenhang mit der hierarchischen Systemzerlegung, für eine hierarchische Überwachung eingesetzt.

3.1 Modifizierte Partikel-Petri-Netze (MPPN)

Nach der Wahl der Petri-Netze als Modellierungswerkzeug und des Partikelfilters als probabilistischer Ansatz, wird in diesem Kapitel das entwickelte Verfahren, in Zusammenhang mit der Definition eines hybriden Systems, dargestellt. Als Grundlage des im weiteren Verlauf der Arbeit vorgestellten Modellierungs- bzw. Überwachungswerkzeuges für hybride Systeme dient die folgende Definition eines hybriden Systems:

Definition 3.1.1: Ein hybrides System ist definiert als ein 5-Tupel $\langle X, U, F, H, O \rangle$:

- X bildet die nicht leere Menge aller zulässigen Zustände des Systems. Diese lassen sich in diskrete $x_D \in X_D$ und kontinuierliche Zustände $x_C \in X_C$ einteilen. Der hybride Systemzustand $x_H \in X$ zum Zeitpunkt k umfasst diskrete und kontinuierliche Zustandsvariablen $x_{H,k} = [x_{D,k}, x_{C,k}]$ und bewegt sich im hybriden Zustandsraum $X = X_D \times X_C$
- U bildet die nicht leere Menge aller zulässigen Eingangsgrößen des Systems. Der hybride Systemeingang $u \in U$ stellt den Vektor aus dem diskreten $u_D \in U_D$ und dem kontinuierlichen $u_C \in U_C$ Eingang dar: $u = [u_{D,k}, u_{C,k}]$
- F ist die Menge der dynamischen Gleichungen zur Beschreibung der Systemdynamik. f stellt die Zustandsübergangsfunktion dar:

$$f : X_D \times X_C \times U_D \times U_C \times \mathbb{R}^m \rightarrow X_D \times X_C$$

$$(x_{D,k+1}, x_{C,k+1}) = f(x_{D,k}, x_{C,k}, u_{D,k}, u_{C,k}, v) \quad v \in \mathfrak{R}^m$$

- H stellt die Menge der Beobachtungsgleichungen dar. Der Zustand des Systems kann durch Messungen O beobachtet werden, so dass gilt:

$$h: X \times \mathfrak{R}^r \rightarrow \mathfrak{R}^p; \quad O \in \mathfrak{R}^p; \quad \eta_t \in \mathfrak{R}^r.$$

- $O = h(x_{H,k}, \eta_k)$ sind hybride Beobachtungen aus dem System.

In dieser Arbeit ist der kontinuierliche Systemanteil durch ein Zustandsraummodell repräsentiert und der diskrete Systemanteil durch ein Petri-Netz. Beide Modellanteile stehen in Wechselwirkung zueinander, sowohl bei der Berechnung der Wahrscheinlichkeitsverteilung über die Zustände als auch durch die speziellen Feuerbarkeitsbedingungen.

3.1.1 Definition der MPPN

Ein MPPN ist definiert als ein 9-Tupel $\langle P, T, Pre, Post, X, F_\pi, F_\delta, \Omega, M_0 \rangle$ und es gilt:

- $P = \{P_1, P_2, \dots, P_n\}$: ist eine endliche nicht leere Menge von Stellen (symbolisch $P^S \subset P$ und numerisch $P^N \subset P$) (Tabelle 3.1.1). Numerische Stellen sind belegt durch Partikel π , und symbolische Stellen durch Konfigurationen δ . $\forall p \in P^N$ ist $m(p)$ die Anzahl von Partikeln $\pi_k \in \Pi_k$ auf der Stelle p zum Zeitpunkt k , wobei Π_k die Menge aller Partikel in den numerischen Stellen ist. Ein Partikel für ein MPPN ist beschrieben durch $\pi_k = [q_k, w_k]$, $q_k \in X_C$ ist der numerische Wert des kontinuierlichen Zustands und $w_k \in [0,1]$ sein Gewicht zum Zeitpunkt k . $\forall p \in P^S$ ist $m(p)$ die Anzahl von symbolischen Token, auch Konfigurationen $\delta_k \in \Delta_k$ genannt, auf der Stelle p , wobei Δ_k die Menge aller Konfigurationen in den symbolischen Stellen ist.
- $T = \{T_1, T_2, \dots, T_m\}$ ist eine endliche nicht leere Menge von Transitionen (numerisch $T^N \subset T$, symbolisch $T^S \subset T$ und hybrid $T^H \subset T$).
- $Pre: P \times T \rightarrow \{0,1\}$ definiert die nicht leere Menge V aller von Stellen p zu Transitionen t gerichteten Kanten, $Pre(p,t) \in V$.
- $Post: P \times T \rightarrow \{0,1\}$ definiert die nicht leere Menge N der von Transitionen zu Stellen gerichteten Kanten, $Post(p,t) \in N$. Wobei $Pre(P_i \times T_j) = 1$ bedeutet, dass es eine Kante von $P_i \rightarrow T_j$ gibt, und $Post(P_i \times T_j) = 1$ dass eine Kante von $T_j \rightarrow P_i$ existiert,

mit $i \in \{1, 2, \dots, n\}$ und $j \in \{1, 2, \dots, m\}$. Die Struktur des Netzes ergibt sich aus der Inzidenzmatrix $C : \forall p \in P$ und $\forall t \in T \ C(p, t) = Post(p, t) - Pre(p, t)$.

Allgemeine Bezeichnung	Graphische Repräsentation
Numerische Stelle	
Symbolische Stelle	
Numerische Transition	
Symbolische Transition	
Hybride Transition	
Gerichtete Kante (Pfeil)	
Konfiguration (symbolische Marke)	
Partikel (numerische Marke)	π

Tabelle 3.1.1: Graphische Elemente der MPPN

- X ist der Zustandsraum des Systems.
- M_0 bezeichnet die Initialmarkierung, mit:

$$\forall p \in P_s, M_0(p) \in \mathbb{N}$$

$$\forall p \in P_N, M_0(p) = \{\pi^i\} \in X_c$$

Die gesamte Markierung des MPPN zum Zeitpunkt k ist $M_k = \{\Delta_k, \Pi_k\}$ (Abbildung 3.1.1):

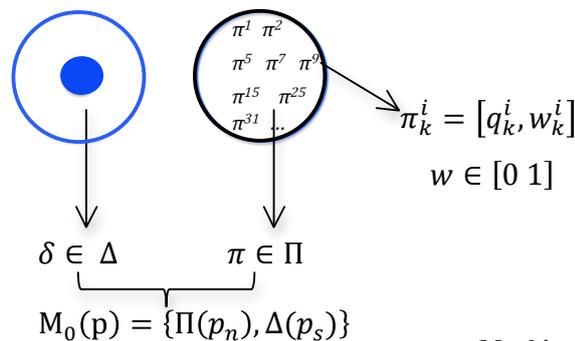


Abbildung 3.1.1: Hybride

Markierung eines MPPN

- $F_\pi : \forall p \in P_N$ ist $F_\pi(p)$ das Differentialgleichungssystem, das einer numerischen Stelle p zugeordnet ist und damit die kontinuierliche Zustandsentwicklung beschreibt. $\forall p \in P_N, F_\pi(p): X_c \times \mathfrak{R}^n \rightarrow X_c$. Die Dynamik der Partikel lässt sich durch $\pi_{k+1} = F_\pi(\pi_k, u_{c,k})$ beschreiben.

- $F_\delta: \forall p \in P^S$ ist $F_\delta(p)$ eine externe Aktion, die einer symbolischen Stelle p eine Konfiguration bzw. mehrere Konfigurationen zuordnet. $\forall p \in P_S, F_\delta(p) \in 2^\zeta$, mit ζ ist die Anzahl der Konfigurationstypen. Die Dynamik der Konfigurationen lässt sich durch $\delta_{k+1} = F_\pi(\delta_k, u_{D,k})$ beschreiben.
- Ω : ist die Menge aller Bedingungen über die Transitionen, wobei Ω_N die numerischen und Ω_S die symbolischen Bedingungen sind.

3.1.2 Dynamik der MPPN

Die Beschreibung der Netzdynamik beruht grundlegend auf der hybriden Markierung der MPPN sowie auf der Aktivierung und Schaltung der Transitionen. Sei ${}^\circ T_j$ die Menge der Eingangsstellen für die Transition T_j , und T_j° die Menge der Ausgangsstellen. Einer numerischen Transition $t_n \in T^N$ sind Bedingungen $\Omega^N(t_n)$ zugeordnet, die sich auf die Zustandsvariablen des Vektors X beziehen. Es gilt: $\Omega^N(t_n)(\pi) = 1$ falls die Bedingungen für die Partikel erfüllt sind, sonst 0. $\Omega^S(t_s)$ beschreibt die Menge der Bedingungen, die einer symbolischen Transition $t_s \in T^S$ zugeordnet sind und welche sich auf ein externes Ereignis beziehen. Es gilt dann $\Omega^S(t_s)(\delta) = 1$, falls die Bedingungen für die Konfiguration erfüllt sind, sonst 0. Einer hybriden Transition $t_h \in T^H$ sind Bedingungen $\Omega^N(t_h)$ bzw. $\Omega^S(t_h)$ zugeordnet, welche sich sowohl auf die kontinuierlichen Zustandsvariablen als auch auf das Auftreten eines externen Ereignisses beziehen. Es gilt für $\forall t_h \in T^H$, $\Omega^N(t_h)(\pi) = 1$ bzw. $\Omega^S(t_h)(\delta) = 1$, falls die Bedingungen für den Partikel bzw. die Konfiguration erfüllt sind, sonst gleich 0.

Regel 3.1.1: Eine Transition T_j ist feuerebar (aktiviert), falls $\forall P \in {}^\circ T$ gilt:

$$m(p) \geq \text{Pr } e(p,t) \quad (3.1.1)$$

Durch das Feuere wird aus einer Markierung $m(p)$ eine neue Markierung $m'(p)$ erzeugt, mit:

$$m'(p) = m(p) - \text{Pr } e(p,t) + \text{Post}(p,t) \quad (3.1.2)$$

Regel 3.1.2: Eine feuerebare symbolische Transition T_j feuert nach den folgenden Regeln:

$$\Pi(p_n) = \Pi(p_n); \quad \Pi(p'_n) = \Pi(p'_n) \cup \Pi(p_n) \quad (3.1.3)$$

$$\Delta(p_s) = \Delta(p_s); \quad \Delta(p'_s) = \Delta(p'_s) \cup \Delta(p_s) \quad (3.1.4)$$

Dadurch ergibt sich aus der alten Markierung $M(p) = \{\Pi(p_n), \Delta(p_s)\}$ eine neue Markierung $M(p') = \{\Pi(p'_n), \Delta(p'_s)\}$ (Abbildung 3.1.2). Das symbolische Feuern verwendet das Konzept des pseudo-Feuerns ([Ric94]-[Fan97]), mit den Token an den symbolischen Stellen (Konfigurationen), d. h. alle Marken (Konfigurationen und Partikel) von den Stellen im Vorbereich werden in die Stellen im Nachbereich herüberkopiert, ohne sie aus den Stellen im Vorbereich zu löschen.

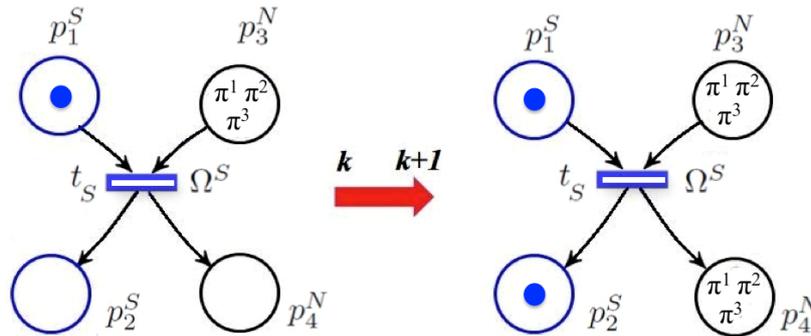


Abbildung 3.1.2: Das Feuern einer symbolischen Transition

Regel 3.1.3: Das Feuern einer feuerbaren numerischen Transition T_j ist definiert wie folgt:

$\forall p_n \in {}^\circ T_j \cap P^N, \forall p'_n \in T_j \circ \cap P^N, \forall p_s \in {}^\circ T_j \cap P^S, \forall p'_s \in T_j \circ \cap P^S$ und $\Pi_F \subseteq \Pi(p_n)$, wobei $\pi \in \Pi_F$, falls $\Omega^N(t_n)(\pi) = 1$:

$$\Pi(p_n) = \Pi(p_n) \setminus \Pi_F \quad ; \quad \Pi(p'_n) = \Pi(p'_n) \cup \Pi_F \quad (3.1.5)$$

$$\Delta(p_s) = \Delta(p_s) \quad ; \quad \Delta(p'_s) = \Delta(p'_s) \cup \Delta(p_s) \quad (3.1.6)$$

Das numerische Feuern verwendet das Konzept des klassischen Feuerns, mit den Partikeln an den numerischen Stellen, d. h. nur die Partikel, die die numerische Bedingung erfüllen, wandern von einer Stelle p_n zu p'_n . Für die symbolischen Konfigurationen gilt das Pseudo-Feuern (Abbildung 3.1.3).

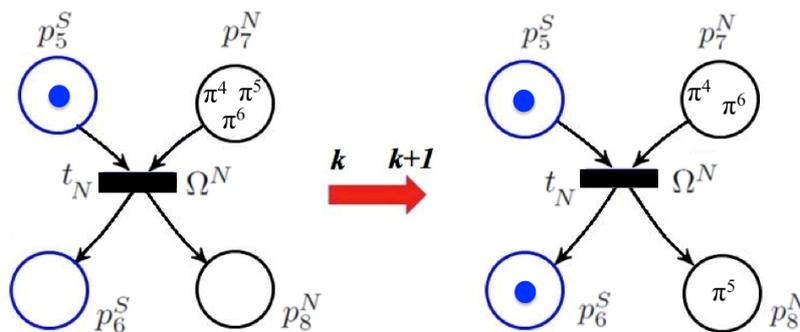


Abbildung 3.1.3: Das Feuern einer numerischen Transition

Regel 3.1.4: Eine hybride Transition T_j ([Ale11]) ist feuerbar, falls $\forall p_i \in {}^\circ T_j$ gilt:

$$m_k(p_i) \geq \text{Pr} e(p_i, T_j), p_i \in P^S \wedge m_k(p_i) \geq 0, p_i \in P^N \quad (3.1.7)$$

Regel 3.1.5: Eine hybride Transition T_j (Abbildung 3.1.4) feuert nach den folgenden Regeln:

$\forall p_n \in {}^\circ T_j \cap P^N, \forall p'_n \in T_j \circ \cap P^N$ und $\Pi_F \subseteq \Pi(p_n)$, wobei $\pi \in \Pi_F$ falls $\Omega_S(t_h)(\delta) = 1$.
Zusätzlich gilt $\forall p_s \in {}^\circ T_j \cap P^S, \forall p'_s \in T_j \circ \cap P^S$ und $\Delta_F \subseteq \Delta(p)$, wobei $\delta \in \Delta_F$ falls $\Omega_S(t_h)(\delta) = 1$:

$$\Pi(p_n) = \Pi(p_n) \setminus \Pi_F \quad ; \quad \Pi(p'_n) = \Pi(p'_n) \cup \Pi_F \quad (3.1.8)$$

$$\Delta(p_s) = \Delta(p_s) \quad ; \quad \Delta(p'_s) = \Delta(p'_s) \cup \Delta(p_s) \quad (3.1.9)$$

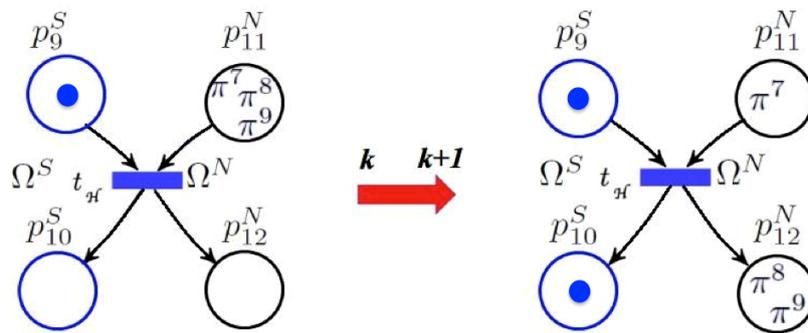


Abbildung 3.1.4: Das Feuereiner hybriden Transition

3.1.3 Gesamtstruktur der MPPN-Prozesse

Zur Schätzung des hybriden Systemzustands verwenden die MPPN das Prinzip der Partikel-Filterung, um sowohl die neuen Werte der Partikel als auch die Folgemarkierung des Petri-Netzes zu schätzen. Diese Schätzung lässt sich in zwei Schritten durchführen: eine Vorhersage (Prediction) und eine Korrektur (Correction). Das Ergebnis ist ein hybrider Zustand (symbolisch-numerisch) und ein kontinuierliches Residuum, die zusammen vom Diagnose-Schritt verwendet werden, um Rückschlüsse auf das Systemverhalten ziehen zu können (Abbildung 3.1.5).

Das Blockdiagramm in Abbildung 3.1.6 zeigt die einzelnen Prozesse der beiden Phasen, die in diesem Kapitel theoretisch behandelt werden und deren Implementierung als Pseudo-Code in Kapitel 5 vorgestellt wird. Die Vorhersage berechnet zunächst die Folgemarkierungen (Prädiktion der Markierung) im Petri-Netz durch das Feuereiner Transitionen (Pseudo-Feuerbarkeit), ausgehend von der aktuellen Markierung zum Zeitpunkt k .

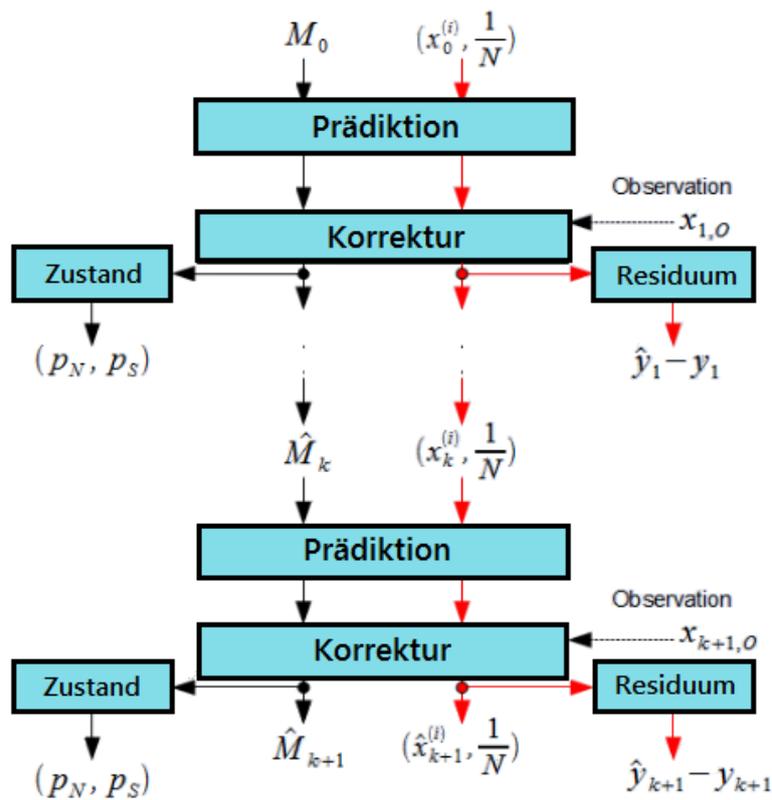


Abbildung 3.1.5: Phasen der Schätzung

Im nächsten Schritt wird die Entwicklung der Partikelwerte, gemäß den Differenzgleichungen, ermittelt (Prädiktion der Partikel). Das Ergebnis dieses Schrittes ist die Menge aller möglichen nächsten Zustände des Systems unter Berücksichtigung sowohl numerischer als auch symbolischer Ungewissheit. Der Korrekturschritt aktualisiert die Vorhersage, basierend auf Beobachtungen aus dem System (Sensordaten). Anhand kontinuierlicher Beobachtungen wird der numerische Zustand korrigiert. Dies erfolgt in drei Schritten: Gewichtung der Partikel, deren Einteilung in Klassen und die Entfernung von irrelevanten Partikeln mittels eines Resampling. Anhand diskreter Beobachtungen wird der symbolische Zustand in Zusammenstellung mit dem numerischen Zustand und durch die Verwendung eines Entscheidungsfindungsverfahrens korrigiert. Nach diesem Korrekturschritt ergibt sich eine neue Markierung des MPPNs zum Zeitpunkt $k + 1$. Die oben beschriebenen Schritte werden in den folgenden Abschnitten näher erläutert.

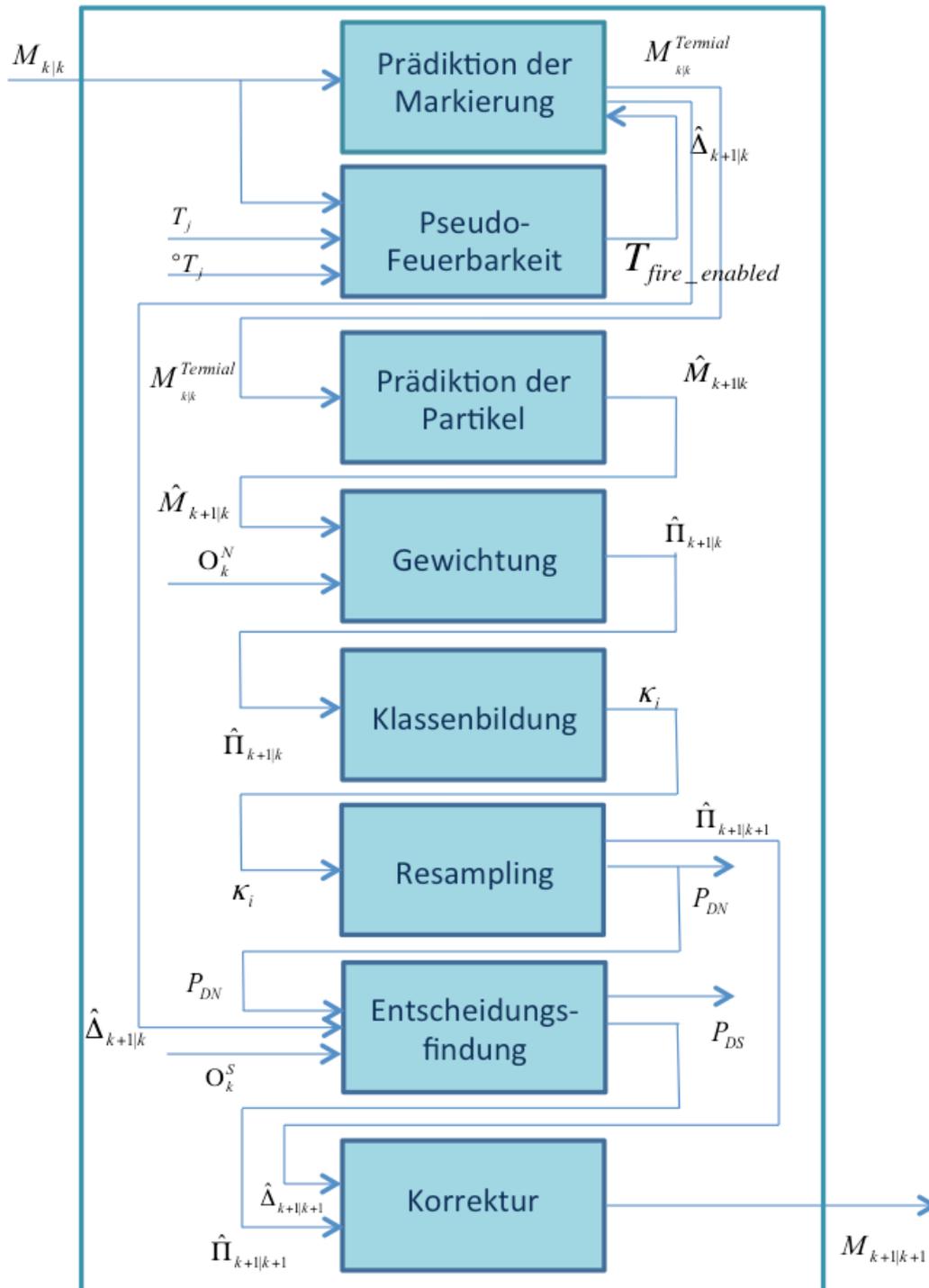


Abbildung 3.1.6: Schätzprozess der MPPN

3.1.4 Prädiktionsschritt im MPPN

Dieser Schritt berücksichtigt beide Teile des Modells und wird sowohl auf die Markierung als auch auf die Partikel angewandt. Für eine bessere Erklärung der Rechenschritte wird das typische Beispiel in der Literatur verwendet, nämlich ein „Thermostat“. Dieses System soll die Temperatur zwischen den Werten T_{\max} und T_{\min} halten, und dementsprechend wird eine

enthält alle möglichen Folgemarkierungen für den Zeitpunkt $k+1$. Eine Transition darf pro Iterationsschritt nur einmal feuern, also existiert kein Zeno-Verhalten (non-Zeno behavior). Ein Modell mit Zeno-Verhalten [Ame05] erlaubt in einem endlichen Zeitintervall eine unendliche Anzahl von Übergängen. Im Fall eines MPPN würde dies bedeuten, dass eine Transition innerhalb eines Zeitschrittes unendlich feuern darf. Im Systemdesign, in der Regelung, in der Steuerung und in der formalen Verifikation werden Zeno-Verhalten ausgeschlossen, da sie nicht mit einem digitalen Regler realisiert sind. In der Wirklichkeit ist es unmöglich, dass ein System einen endlosen Zustandswechsel in endlicher Zeit verfolgt. Ein interessantes Beispiel zu hybriden Systemen mit Zeno Verhalten zeigt der springende Ball. Jedes Mal, wenn der Ball aufprallt, verliert er Energie, so dass die nachfolgenden Ballsprünge zeitlich immer näher zusammenrücken. In der Tat kann man ohne die Berücksichtigung von Kräften den springenden Ball nicht korrekt definieren, und das Modell wäre, von einem mechanischen Standpunkt aus betrachtet, sinnlos.

Abbildung 3.1.7 zeigt eine Initialmarkierung $M_0 = (p_0, p_1, \text{on}, \text{off}, p_2) = (20p, 0, 1, 0, 0)$, also 20 Partikel an der numerischen Stelle „P0“ und eine Konfiguration an der symbolischen Stelle „on“. Nach dem Simulationsschritt „#19“ (Abbildung 3.1.8) ist die Markierung $M_{19} = (0p, 11p, 1, 0, 9p)$. Daraus wird anhand der Feuerbarkeitsregeln die Markierung für den nächsten Schritt „#20“ berechnet: T_1 feuert, 1 Partikel (1p) aus „P1“ erfüllt die Bedingung an T_1 , deswegen wandert er nach „P0“. T_2 feuert (System wird ausgeschaltet), somit wird die symbolische Stelle „off==1“.

```

Marking after simulation step #19
(0 11p 1 0 9p )

.....
This is Simstep #20

Transition is fired 1
(1p 10p 1 0 9p )

Transition is fired 2
(1p 10p 1 1 19p )

Transition is fired 3
(1p 10p 1 1 20p )

Transition is fired 4
(1p 30p 1 1 20p )

The Terminal Marking is :
(1p 30p 1 1 20p )

```

Abbildung 3.1.8: Berechnung der Endmarkierung

Bei T_2 handelt es sich um eine symbolische Transition, bei der das Konzept des Pseudofeuerns nach (3.1.3) und (3.1.4) angewendet wird. Diese bewirkt so, dass die

Konfiguration „on“ nach „off“ nur kopiert wird („on==1“ bleibt aktiv) und die 10p von „P1“ nach „P2“ kopiert werden. Es ergibt sich eine Zwischenmarkierung (1p, 10p, 1, 1, 19p). Durch das Feuern von T_3 wird der 1p aus „P0“ in „P2“ kopiert; somit ergibt sich (1p, 10p, 1, 1, 20p). Durch das Feuern von T_4 werden die 20p von „P2“ in „P1“ kopiert (1p, 30p, 1, 1, 20p). An dieser Stelle befindet sich das System in all seinen möglichen erreichbaren Zuständen. Diese resultierende Markierung stellt die Endmarkierung $M_{20}^T = (1p, 30p, 1, 1, 20p)$ dar.

b) Prädiktion der Partikel:

In diesem Schritt entwickeln sich die wertkontinuierlichen Zustände $q_k^i \in X$ der einzelnen Partikel $\pi_{k+1|k}^i = [q_{k+1|k}^i, w_{k+1|k}^i]$ gemäß den differentialen Gleichungen in den numerischen Stellen (Abbildung 3.1.9):

$$q_{k+1}^i = F(p, k, q_k^i) + v(p), \quad \forall p \in P_N, \forall \pi \in \hat{\Pi}_{k+1|k}(p), \quad i = 1 \dots N \quad (3.1.11)$$

N ist die Anzahl der Partikel des Iterationschrittes und $\hat{\Pi}_{k+1|k}(p)$ ist der numerische Teil der geschätzten Markierung $\hat{M}_{k+1|k}$. Der Zweck dieses Schrittes ist, den Erwartungswert $E(x)$ der Zufallsgröße x mit der Dichtefunktion $f(t)$ zu berechnen:

$$E(x) = \int_{-\infty}^{+\infty} t' f(t') dt' \quad (3.1.12)$$

Für N unabhängige Zufallsvariablen q_i ist der arithmetische Mittelwert:

$$\bar{q} = \frac{1}{N} \sum_{i=1}^N q_i \quad (3.1.13)$$

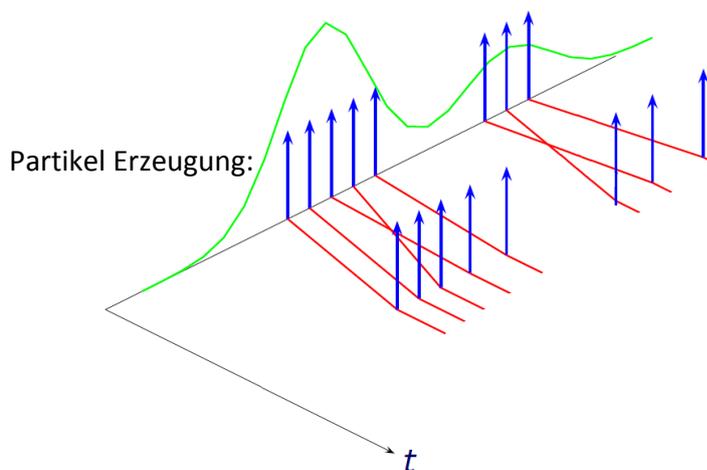


Abbildung 3.1.9: Prädiktion der Partikel

Werden die Partikel auf die numerischen Stellen normalverteilt $\mathcal{N}(\cdot)$, handelt es sich um eine Gaussmarkierung M_G . Diese lässt sich als eine aleatorische Variable mit dem normalen Mittelwert $\hat{x}(p)$ an der numerischen Stelle p , der Kovarianz $\mathcal{K}_k(p)$ und einem Gewicht $w_k(p)$ definieren:

$$\Pr(M_G(k, p)) = w_k(p) \mathcal{N}(\hat{x}(p), \mathcal{K}_k(p)) \quad (3.1.14)$$

3.1.5 Korrekturschritt im MPPN

Korrektur: Nach einer Prädiktion erfolgt eine Korrekturphase der geschätzten Markierung sowie der Zustandswerte.

a) Numerische Korrektur: Diese Phase besteht aus drei Schritten:

Die Gewichtung der Partikel: Die Partikel $\pi_{k+1|k}^i = [q_{k+1|k}^i, w_{k+1|k}^i]$ werden, wie bei der Partikelfilterung, durch eine Wahrscheinlichkeitsdichtefunktion $\Pr(q_k, O_k^N, \mathbf{v}_k^O)$, die von der Systembeobachtung (Sensordaten) O_k^N und einem Rauschen \mathbf{v}_k^O abhängt, gewichtet (Abbildung 3.1.10):

$$\tilde{w}_{k+1}^i = \Pr(q_{k+1}^i, O_k^N, \mathbf{v}_k^O), \quad (3.1.15)$$

\tilde{w}_{k+1}^i ist der likelihood Wert des i-ten Partikels.

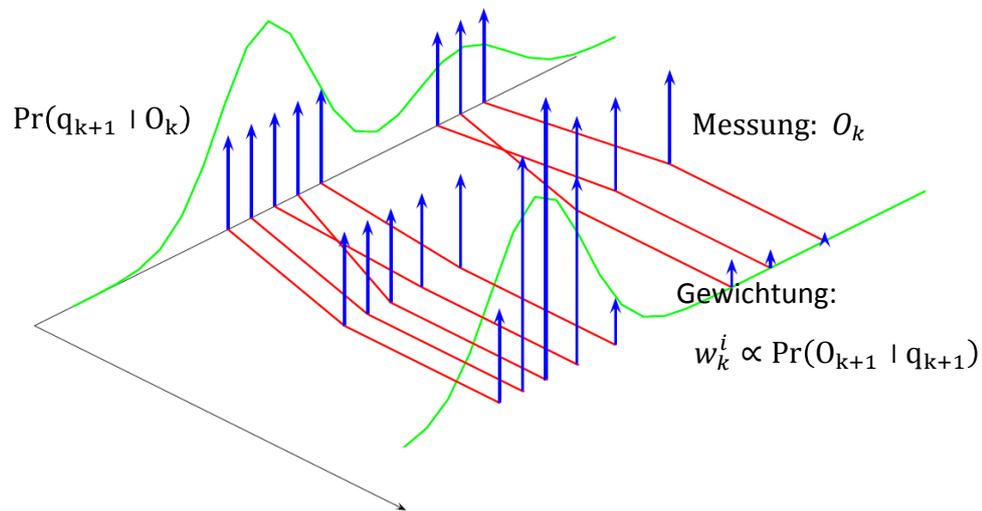


Abbildung 3.1.10: Korrektur der Prädiktion

Anschließend werden die Gewichte \tilde{w}_{k+1}^i über die Summe der Gewichte aller Partikel

normiert, so dass $\sum_{i=1}^J w_{k+1}^i = 1$ ist. Es gilt:

$$w_{k+1}^i = \frac{\tilde{w}_{k+1}^i}{\sum_{i=1}^J \tilde{w}_{k+1}^i}, \tag{3.1.16}$$

wobei J die Anzahl aller Partikel ist und w_{k+1}^i die normierte Gewichtung.

Die Wahrscheinlichkeitsverteilung eines numerischen Zustandsvektors an eine numerische Stelle $\forall p \in P_N$ ergibt sich aus der Summe der Gewichte in dieser Stelle:

$$\forall p \in P_N, P_{DN} = P(\pi_k \in \widehat{M_{k+1|k}(p)}) = \sum_{\pi_{k+1|k}^i \in \widehat{M_{k+1|k}(p)}} w_k^i \tag{3.1.17}$$

Die Klassenbildung: Für eine robuste numerische Korrektur werden aus den gewichteten Partikeln Klassen gebildet (Abbildung 3.1.11). Hierbei werden die Partikel im ganzen Netz und, unabhängig von ihrer Lage nur nach ihren Gewichten absteigend in einer Liste sortiert.

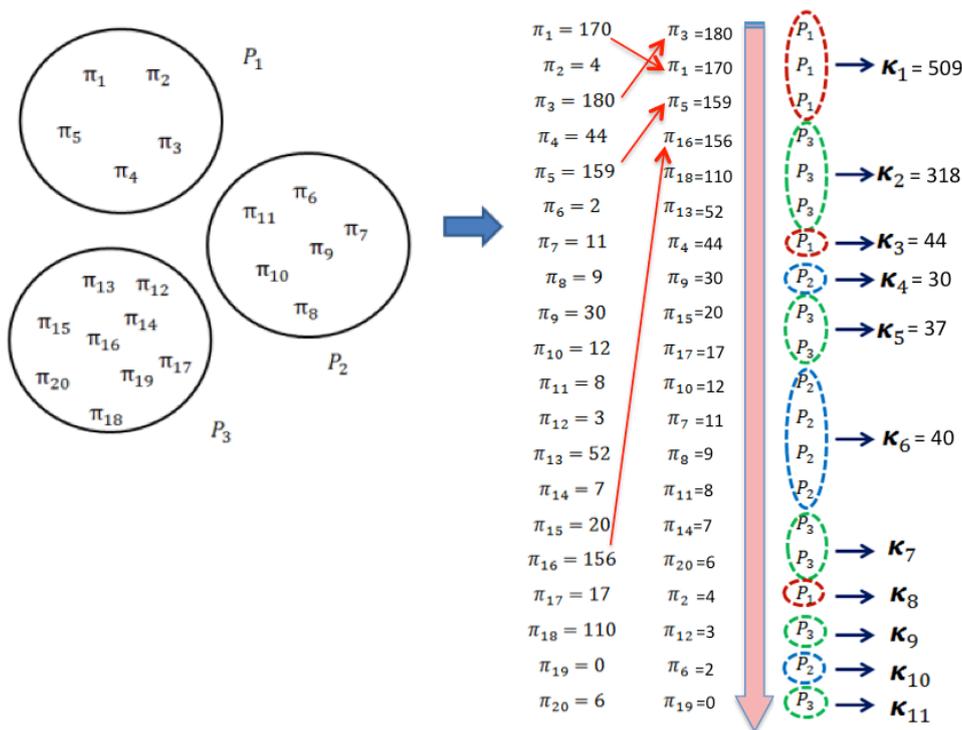


Abbildung 3.1.11: Gruppierung der Partikel und Klassenbildung

Ausgehend vom Partikel mit der höchsten Gewichtung, wird eine Klasse wie folgt definiert:

Eine Klasse κ ist die Menge aller aufeinanderfolgenden Partikel in der Liste L , die sich dieselbe Stelle teilen und nicht die Menge aller Partikel, die sich in derselben numerischen Stelle befinden. Damit ist κ_1 die Klasse, welche den Partikel mit der

höchsten Gewichtung enthält. Das bedeutet, seien π^i und π^j zwei aufeinanderfolgende Elemente der Liste L , d. h. $w(\pi^i) > w(\pi^j)$, dann gehören π_i und π_j zur selben Klasse, wenn sie die gleiche numerische Stelle $p \in P_N$ besitzen, d. h. $p(\pi^i) = p(\pi^j)$ markieren. Dieser Schritt der Klassenbildung erleichtert die Entscheidung über den höchstwahrscheinlichen numerischen Zustand wenn die Wahrscheinlichkeiten an den numerischen Stellen nahe beieinander liegen. Somit stellt die numerische Stelle mit der hochgewichteten Klasse $p(\kappa_1)$ den höchstwahrscheinlichen numerischen Zustand dar.

Nach dem Erhalt einer Temperaturmessung von z. B. 21,5°C, einer Gewichtung der Partikel und einer Klassenbildung lassen sich die Wahrscheinlichkeiten der einzelnen Zustände P0 („Heating“), P1 („Cooling_On“) und P2 (Cooling_Off) berechnen (Abbildung 3.1.12). In diesem Simulationsschritt ist das System höchstwahrscheinlich eingeschaltet („on“: 61,48%) und hat den Betriebsmodus Kühlen („P1“: 59,24%). Die Berechnung der Wahrscheinlichkeiten für die symbolischen Stellen „on“ und „off“ wird in einem späteren Abschnitt bei der symbolischen Korrektur näher erläutert.

```
The Observation is: 21.5
Probability 'P0' : 2.23204%
Probability 'P1' : 59.2493%
Probability 'P2' : 38.5186%
.....
Probability 'on' : 61.4814%
Probability 'off' : 38.5186%
```

Abbildung 3.1.12: Wahrscheinlichkeiten an numerischen und symbolischen Stellen

Das Resampling: Beim Resampling werden die Partikel mit sehr kleiner Gewichtung gelöscht. Die Anzahl der Partikel nach dem Resampling ist dabei genauso groß wie bei der Initialisierung. Das bedeutet, aus den N ungleich gewichteten Partikeln werden wieder N gleichgewichtete Partikel erzeugt (Abbildung 3.1.13).

Count	Place	Value	Weight	ID	Type
0	2	1	0.000000	0	Token
1	1	19.9656	0.050000	1	Particle
2	4	19.9135	0.050000	2	Particle
3	1	20.5697	0.050000	3	Particle
4	1	19.9263	0.050000	4	Particle
5	4	20.4863	0.050000	5	Particle
6	4	20.3791	0.050000	6	Particle
7	4	19.6742	0.050000	7	Particle
8	1	19.7276	0.050000	8	Particle
9	1	19.7486	0.050000	9	Particle
10	1	19.8313	0.050000	10	Particle
11	4	19.8969	0.050000	11	Particle
12	1	19.7276	0.050000	12	Particle
13	1	20.2865	0.050000	13	Particle
14	4	20.4863	0.050000	14	Particle
15	1	20.4888	0.050000	15	Particle
16	4	20.7037	0.050000	16	Particle
17	1	19.7996	0.050000	17	Particle
18	1	19.2929	0.050000	18	Particle
19	4	20.7037	0.050000	19	Particle
20	1	19.5551	0.050000	20	Particle

Average Value: 20.0582

Residuum: 1.44184

Marking after simulation step #20
(0 12p 1 0 8p)

Abbildung 3.1.13: Ergebnisse eines Simulationsschrittes

Falls zu viele Partikel beim Resampling herausgefiltert wurden, müssen die verbliebenen Partikel kopiert werden. $\hat{\Pi}_{k+1|k} \subseteq \hat{M}_{k+1|k}$ bezeichnet die sortierten Partikel, und $\hat{\Pi}_{k+1|k+1}$ sind die Partikel nach dem Resampling (Abbildung 3.1.14).

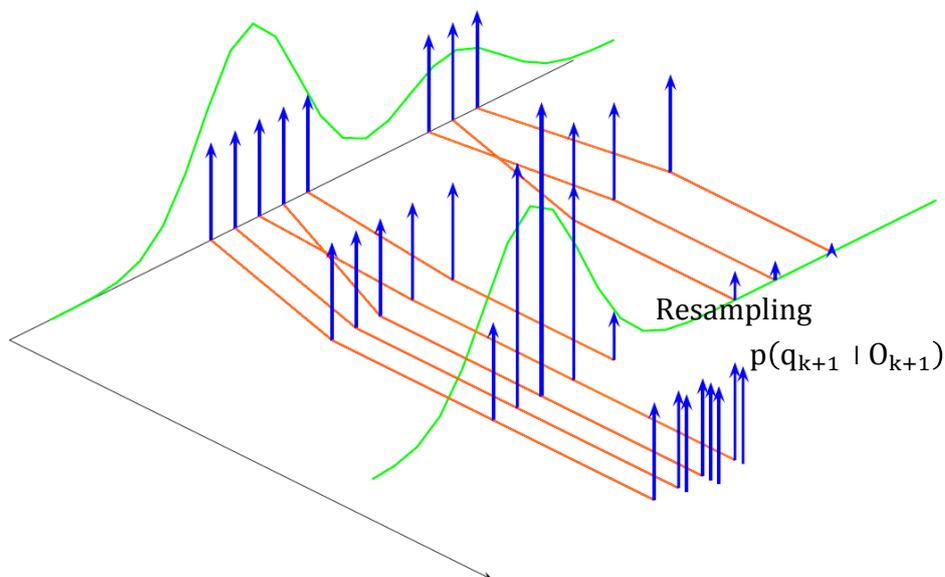


Abbildung 3.1.14: Resampling

b) Die symbolische Korrektur

Dieser Korrekturschritt aktualisiert die symbolische Markierung des Petri-Netzes aus einer Beobachtung der Konfiguration, um die Markierung $\hat{M}_{k+1|k+1}$, welche den Systemzustand zum Zeitpunkt $k + 1$ repräsentiert, zu ermitteln. Der erste Schritt in diesem Prozess besteht darin, die Wahrscheinlichkeitsverteilung über den symbolischen Zuständen P_{DS} , in Abhängigkeit von der numerischen Verteilung P_{DN} , zu berechnen. Diese Abhängigkeit und die Berücksichtigung des Einflusses der beiden Teile ist ein weiteres Merkmal, das die MPPN charakterisiert. Um den Einfluss des numerischen Teils betrachten zu können, definieren wir einen Zustandsgraphen G_S :

$\forall p_n \in P^N, \forall p_s \in P^S$ ist

$$G_S = \begin{bmatrix} S_{1,1} & \dots & S_{1,K} \\ \vdots & \ddots & \vdots \\ S_{N,1} & \dots & S_{N,K} \end{bmatrix}, \quad S_{i,j} = \begin{cases} 1 & (p_n^i, p_s^j) \in R_G \\ 0 & \text{sonst} \end{cases} \quad (3.1.18)$$

mit R_G ist der Erreichbarkeitsgraph, (p_n^i, p_s^j) ist der hybride Zustand, gegeben durch die i -te numerische Stelle und die j -te symbolische Stelle des Petri-Netzes. S sagt aus, ob der hybride Zustand erreichbar ist ($S = 1$) oder nicht ($S = 0$). Die Wahrscheinlichkeitsverteilung über den symbolischen Zustandsvektor wird wie folgt berechnet:

$$P_{DS}(\hat{\Delta}_{k+1|k}(p_s)) = \sum_{n=1}^K G_S(i, j) P_{DN}(\hat{\Pi}_{k+1|k}(p_n)), \forall p_n \in P^N, \forall p_s \in P^S, \quad (3.1.19)$$

Die Stelle mit der höchsten Wahrscheinlichkeit P_{DS} wird mit p_m bezeichnet. $\Omega(p_m)$ repräsentiert die Menge der Konfigurationen, die in p_m enthalten sind. Um die Vorhersage des symbolischen Zustands zu aktualisieren, wird ein Entscheidungsmechanismus benötigt. Dieser vergleicht die symbolische Beobachtung O_k^S mit $\Omega(p_m)$ und überprüft die Erreichbarkeit des hybriden Zustands $(p(\kappa_1), p_m)$, wobei $p(\kappa_1)$ die Klasse mit der höchsten Gewichtung ist. Das Ergebnis dieses Entscheidungsalgorithmus ist die geschätzte symbolische Markierung $\hat{\Delta}_{k+1|k+1}$ zum Zeitpunkt $k + 1$, welche entweder der Markierung des höchstwahrscheinlichen Zustands $\hat{\Delta}_{k+1|k+1} = \hat{M}_{k+1|k}(p_m)$ oder dem vorherigen Zustand $\hat{\Delta}_{k+1|k+1} = \hat{\Delta}_{k|k}$ entsprechen könnte.

Die aus dem Korrekturprozess resultierende Markierung $\hat{M}_{k+1|k+1} = \hat{\Pi}_{k+1|k+1} \cup \hat{\Delta}_{k+1|k+1}$ stellt den hybriden Systemzustand zum Zeitpunkt $k+1$ dar.

3.2 Fehlerdiagnose

Im Hinblick auf eine optimale Fehlererkennung und Unterscheidung zwischen den Fehlerklassen in der Fehlerdiagnose ist eine Betrachtung des Residuums und der Konsistenz der Zustände erforderlich. Gegenüber klassischen Beobachtungs- und Diagnoseverfahren für hybride Systeme basiert der hier entwickelte Fehlererkennungsmechanismus auf einer simultanen Zusammenstellung des Schätzwertes für den wertkontinuierlichen Systemzustand und die Erreichbarkeit des diskreten Zustands. Somit ist die Residuenberechnung durch eine Konsistenzanalyse bzw. eine Erreichbarkeitsanalyse des hybriden Systemzustands ergänzt. Diese gemeinsame Betrachtung beider Teile des Modells - numerisch und symbolisch - von der Schätzungsphase über die Berechnung der Wahrscheinlichkeitsverteilungen P_{DN} und P_{DS} bis zur Fehlererkennung - ermöglicht eine Unterscheidung der Ursachen bei einer Abweichung im Systemverhalten und damit eine Einordnung der Fehler. Abweichungen können aufgrund einer stochastischen Störung im Normalbetrieb (externes Rauschen), eines funktionalen Fehlverhaltens des Systems (interner Komponentenfehler, z. B. Sensorausfall) oder einer globalen Verhaltensinkonsistenz (z. B. Modellfehler) entstehen.

Für die Bewertung der Abweichungen und deren Zuordnung in eine der obigen Fehlerklassen werden folgende Analyseschritte durchgeführt:

a) Analyse der kontinuierlichen Dynamik

Zur Detektion der Abweichungen in den kontinuierlichen Systemvariablen werden eine Residuumerzeugung und eine Residuenauswertung durchgeführt:

- **Residuumerzeugung:** Die Grundidee dieses Schrittes ist der Vergleich von realen Daten m (aus der Messung) mit Referenzdaten p , die anhand eines Modells aus der Spezifikation sp abgeleitet werden (Abbildung 3.2.1). Es ergibt sich ein Residuum $\varepsilon_i = m - p$ des kontinuierlichen Zustandsvektors, das in einem nächsten Schritt ausgewertet wird.

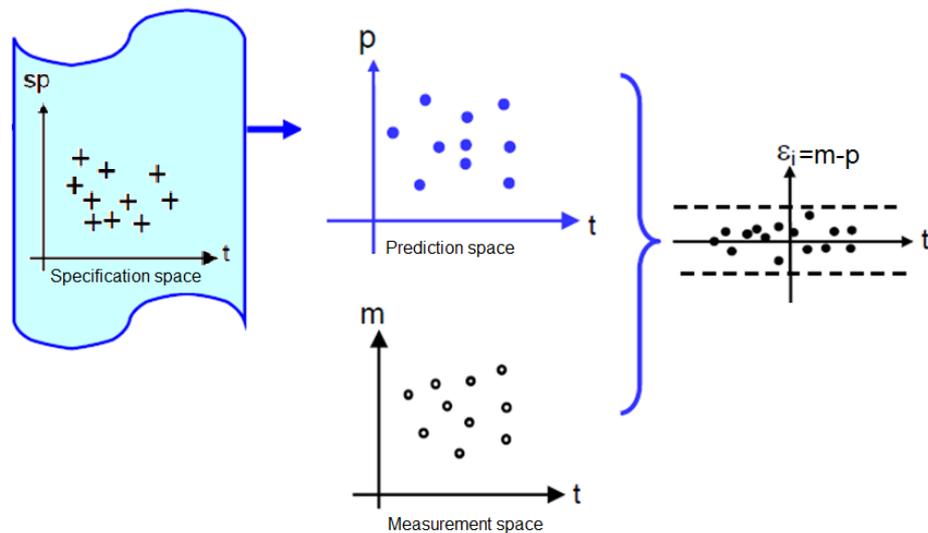


Abbildung 3.2.1: Residuungenerierung

- **Residuenauswertung:** Sowohl im Fehlerfall als auch im fehlerfreien Fall (aber in Anwesenheit von Messrauschen oder Modellungenauigkeit) weichen die Residuen von null ab, deswegen werden sie mittels einer Entscheidungslogik, z. B. durch den Vergleich des Residuums mit einer Schwelle, ausgewertet. Je nach verfügbaren Sensorsignalen und der Struktur des Fehlerdiagnose-Systems sind verschiedene Stufen der Residuenauswertung möglich. In diesem Überwachungskonzept wird zwischen zwei Arten von Residuen bzw. Schwellen unterschieden: eine Kontrollschwelle (S_K), die Abweichungen detektiert, aber keinen Fehler meldet und eine Erkennungsschwelle (S_E), die sehr wohl auf Abweichungen reagieren soll und das Auftreten von Fehlern meldet.

b) Analyse der diskreten Dynamik

Für die Konsistenzanalyse wird der numerische Zustand, der die Klasse mit der höchsten Gewichtung enthält $P(\kappa_1)$, in Kombination mit dem höchstwahrscheinlichen symbolischen Zustand p_m auf die Erreichbarkeit überprüft. Dieser Schritt ist notwendig, da sich die Klasse mit der höchsten Gewichtung nicht unbedingt im höchstwahrscheinlichen numerischen Zustand befindet und damit nicht unbedingt in Kombination mit dem symbolischen Zustand eine erreichbare Markierung bildet. Ist das Ergebnis dieser Sonderprüfung der Erreichbarkeit positiv, spricht man von einer Konsistenz, ansonsten von einer Inkonsistenz innerhalb des symbolischen Teils bzw. von einer Konfliktsituation im Systemverhalten.

$$(P(\kappa_1), p_m) = \begin{cases} \text{Konsistent} & \text{für } (P(\kappa_1), p_m) \in R_G \\ \text{inconsistent} & \text{sonst} \end{cases} \quad (3.2.1)$$

Nach der Residuenauswertung und Konsistenzanalyse kann eine Fehleranalyse durchgeführt werden, um die Ursache des Fehlers zu ermitteln. In dieser Arbeit ist die Fehlerdiagnose bis zur Fehleranalyse und Alarmauslösung implementiert (Abbildung 3.2.2). Die Rekonfiguration ist nicht Bestandteil dieser Arbeit.

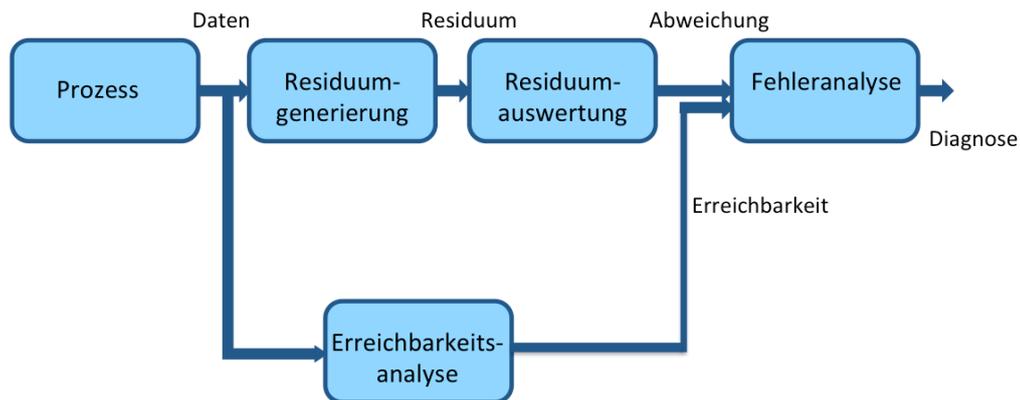


Abbildung 3.2.2: Schematische Darstellung des entwickelten Fehlerdiagnosemechanismus

Ein Überwachungsansatz, der auf Zustandsschätzung basiert, erfordert eine Modellierung der Unsicherheit, einerseits über die Werte der Zustände (maximale zulässige Abweichung aus der Spezifikation s_p), andererseits über die Beobachtungen (Messung m) (Abbildung 3.2.3). In MPPN werden beide Aspekte sowohl numerisch als auch symbolisch betrachtet:

- **Unsicherheit über die Zustände:** Numerisch wird diese Unsicherheit durch die Verwendung eines numerischen Filters, hier der Partikel-Filter, modelliert. Dies berücksichtigt das Prozessrauschen in der Prädiktionsphase und das Messrauschen in der Korrekturphase. Die Unsicherheit über die symbolischen Zustände wird durch die Makro-Markierung dargestellt. Diese entspricht dem Vorhandensein von Token an verschiedenen Orten zum gleichen Zeitpunkt, wobei der tatsächliche Systemzustand nur einen Teil dieser Markierung bildet.
- **Unsicherheit der Beobachtung:** Diese wird für numerische sowie symbolische Sensordaten modelliert, indem die gelieferte Messung in einem Intervall zugelassen wird. Dieses Intervall stellt die Nebenbedingung über die Genauigkeit des Systems dar, welche in der Spezifikation als Messrauschen (Abbildung 3.2.3 (b)) eingegeben wird.

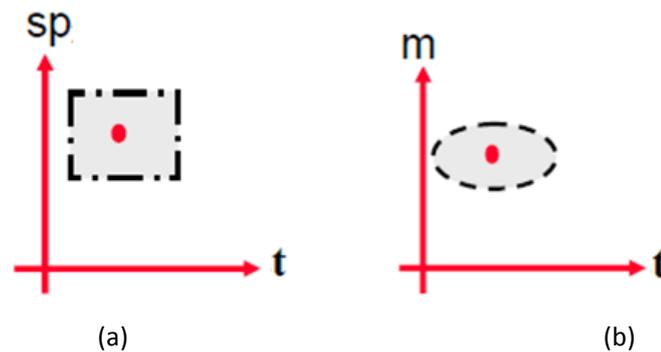


Abbildung 3.2.3: Modellierung der Unsicherheit durch Prozessrauschen (a) (aus der Spezifikation sp) und Messrauschen (b) (aus der Messung m)

3.3 Architektur des Ansatzes

Die generische Architektur des neuen Überwachungsansatzes ist in Abbildung 3.3.1 dargestellt und besteht im Wesentlichen aus zwei Komponenten:

1) Petri-Netz Block

Die ereignisdiskrete Systemdynamik wird durch ein Petri-Netz mit n_D Stellen nachgebildet, dessen Markierungsvektor M den diskreten Zustand x_D repräsentiert. Somit bezeichnet $x_D(0) = M_0$ den diskreten Anfangszustand und damit die Anfangsmarkierung. Alle möglichen Markierungsvektoren des Petri-Netzes sind in der endlichen Menge X_D zusammengefasst. Die Zustandsübergänge werden durch externe oder interne Ereignisse ausgelöst. Bei einem externen Ereignis wird der Schaltvorgang unmittelbar durch die Änderung von diskreten Eingangsgrößen u_D ausgelöst und führt zu einer neuen Markenbelegung der symbolischen Stellen P_S . Die Markierung dieser Stellen stellt den symbolischen diskreten Zustand x_{DS} dar. Im Fall eines internen Ereignisses führt die Änderung der internen Systemdynamik (z. B. Grenzwertüberschreitung) zu einem Schalten der numerischen Zustände x_{DN} . Diese werden durch numerische Stellen P_N , welche Differentialgleichungen enthalten und damit die kontinuierliche Dynamik des Systems durch die Entwicklung der Partikel beschreiben, dargestellt.

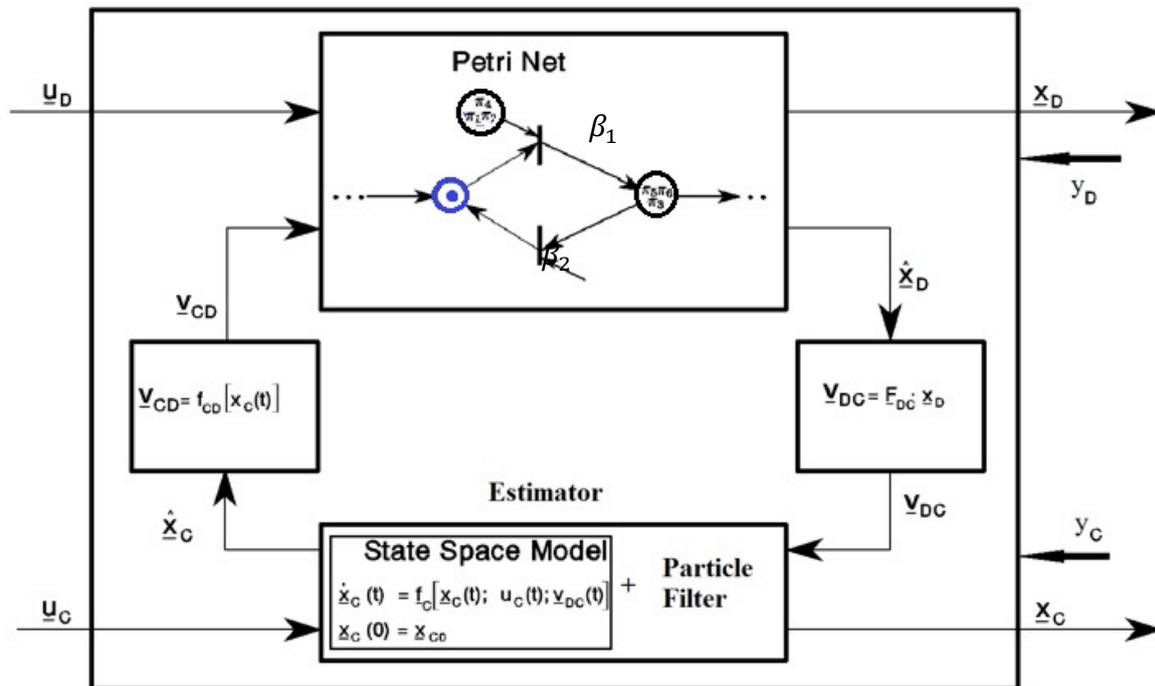


Abbildung 3.3.1: Schematische Darstellung der Wechselwirkungen im MPPN

Die gesamte Petri-Netz-Markierung besteht aus den Markierungen beider Modellteile und stellt damit einen möglichen diskreten Zustand $x_D = [x_{DN}, x_{DS}]$ des Systems dar. Der diskrete Folgezustand $x_D(t+1)$ ergibt sich aus den aktuellen Stellenbelegungen $x_D(t)$ des Petri-Netzes, den algebraischen oder logischen oder binären Verknüpfungen an den Schaltbedingungen β_i , an den Transitionen t_i , den diskreten Eingangsgrößen u_D und aus dem Ausgangsvektor v_{CD} des kontinuierlich/diskreten Interfaces:

$$x_D(t+1) = f_D(x_D(t), u_D(t), v_{CD}(t), \beta_i(t))$$

Der Ausgangsvektor v_{DC} des diskret/kontinuierlichen Interfaces stellt dem kontinuierlichen Modellteil (der Estimator) die höchstwahrscheinlichste Markierung zur Verfügung:

$$v_{DC}(t) = F_{DC} \cdot x_D(t)$$

F_{DC} ist eine Matrix aus den Elementen $\{0,1\}$, die dadurch festlegt, welche numerischen Stellen vom Estimator weiter verwendet werden, um die Werte der Partikel π_i nach den entsprechenden Differentialgleichungen zu berechnen.

2) Ein Estimator-Block

Durch die Verwendung der Differenzgleichungen f_C des kontinuierlichen Systemeingangs u_C bzw. -ausgangs y_C und des Prinzips der Partikelfilterung wird der

kontinuierliche Zustandsvektor \hat{x}_C geschätzt. Dieser ist auch von der Wechselwirkung mit dem diskreten Teil über \underline{V}_{DC} abhängig. Die kontinuierliche Dynamik des Systems ist durch die Entwicklung des kontinuierlichen Zustandsvektors \underline{x}_C nach einem System von n -Differentialgleichungen beschrieben:

$$\dot{x}_C(t) = f_C(\underline{x}_C(t), \underline{V}_{DC}(t), \underline{u}_C(t)).$$

Der kontinuierliche Modellanteil wirkt auf den ereignisdiskreten Modellanteil über das \underline{V}_{CD} ein und stellt, in Abhängigkeit vom kontinuierlichen Zustand \underline{x}_C und von den kontinuierlichen Eingangsgrößen, den booleschen Vektor zur Verfügung:

$$\underline{V}_{CD}(t) = f_{CD}(\hat{x}_C(t))$$

Diese Schnittstelle wirkt nur auf bestimmte diskrete Zustände, nämlich diejenigen, die eine Grenzwertüberschreitung im kontinuierlichen Teil modellieren.

Eine Schätzung des hybriden Systemzustands ergibt sich also aus der Schätzung des kontinuierlichen \hat{x}_C und des diskreten Zustands \hat{x}_D , unter Berücksichtigung der simultanen Interaktion zwischen den beiden. Das kontinuierliche Teilmodell berechnet aus der geschätzten Markierung der numerischen Stellen P_N und deren kontinuierlichen Werten eine Wahrscheinlichkeitsverteilung P_{DN} . Das diskrete Teilmodell berechnet aus der geschätzten Markierung der symbolischen Stellen P_S und deren Tokenbelegung eine Wahrscheinlichkeitsverteilung P_{DS} . Der höchstwahrscheinlichste hybride Systemzustand ergibt sich aus der erreichbaren Stellenbelegung mit der höchsten Wahrscheinlichkeit.

3.4 Hierarchische Überwachung mittels MPPN

Ein wichtiges Merkmal von dynamischen Systemen ist die *Komplexität*. Man unterscheidet zwischen drei unterschiedlichen Dimensionen von Komplexität: Zeit-, Raum- und Strukturkomplexität. In der Informatik befasst man sich mit der Komplexität der Algorithmen in Bezug auf die Zeit (Anzahl der benötigten Rechenschritte) und den Raum (Speicherbedarf). Die Komplexität wird aber, im Hinblick auf den Systementwurf, in Bezug auf die Struktur betrachtet. Die strukturelle Komplexität von dynamischen Systemen ergibt sich aus der hohen Anzahl der Komponenten sowie deren Abhängigkeiten oder Wechselwirkungen untereinander. Die Reduktion der Komplexität durch die Zerlegung des Systems in kleinere Subsysteme erfordert die Berücksichtigung einiger Modellierungsaspekte, wie z. B. die Zerlegungsfähigkeit, die hierarchische Darstellung und die hybride Modellierung.

In einer hierarchischen Zerlegung sind die Subsysteme so angeordnet, dass jede Ebene nur auf hierarchisch niedriger stehende Ebenen zugreifen kann und keine Kenntnisse über höher stehende hat. Das Modell des Systems besteht dann aus den Modellen dieser Ebenen. Der globale Systemzustand setzt sich aus den Zuständen der Teilsysteme zusammen, die so genannten lokalen Zustände. Durch die Kombination dieser lokalen Zustände $\{z_i\}$ wächst der Zustandsraum des globalen Zustands $Z = \{z_1, z_2, \dots, z_N\}^T$ schnell an. Die Modellierung wird somit komplexer und der Rechenaufwand höher. Eine erfolgreiche Modellierung sollte die Komplexität der Beschreibung begrenzen. Dies wird durch eine Zerlegung des Systems in Teilsysteme erreicht. Für solch eine Zerlegung wird das Überwachungskonzept durch hierarchische Petri-Netze und eine hierarchische Zustandsraummodellierung realisiert (Abbildung 3.4.1).

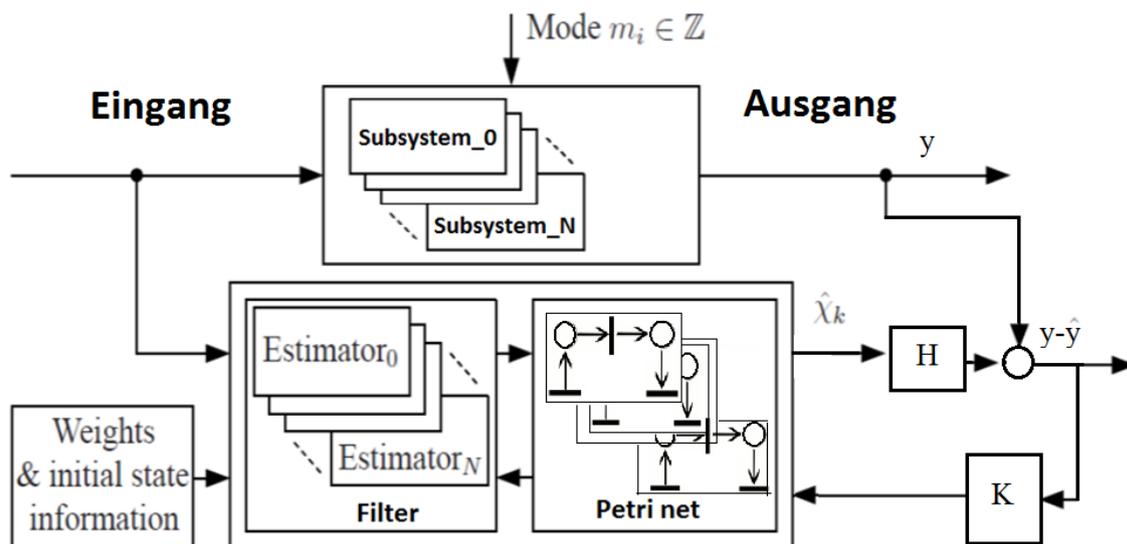


Abbildung 3.4.1: Hierarchische Überwachung mittels MPPN

Es handelt sich dann um ein N-Stufen Überwachungssystem ([Mek09a]-[Mek10]) (Abbildung 3.4.2). Der Vorteil dieses Ansatzes ist die reduzierte Ordnung des Zustandsraummodells. Denn unabhängig von der Anzahl der Verhaltensebenen lässt sich der Zustandsvektor einer beliebigen Ebene nur um den geschätzten Zustand der vorherigen Ebene erweitern:

$$u'_1 = [u_1]; \quad u'_2 = \begin{bmatrix} \hat{x}_1 \\ u_2 \end{bmatrix}; \quad \dots; \quad u'_3 = \begin{bmatrix} \hat{x}_2 \\ u_3 \end{bmatrix}; \quad u'_N = \begin{bmatrix} \hat{x}_{N-1} \\ u_N \end{bmatrix}$$

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, u_1); \quad \dot{x}_2 = f_2(\hat{x}_1, x_2, u_2); \quad \dots; \quad \dot{x}_N = f_N(\hat{x}_{N-1}, x_N, u_N) + w \\ y_1 &= h(x_1) \quad ; \quad y_2 = h(x_1, x_2) \quad ; \quad \dots; \quad y_N = h_N(x_{N-1}, x_N) + v \end{aligned}$$

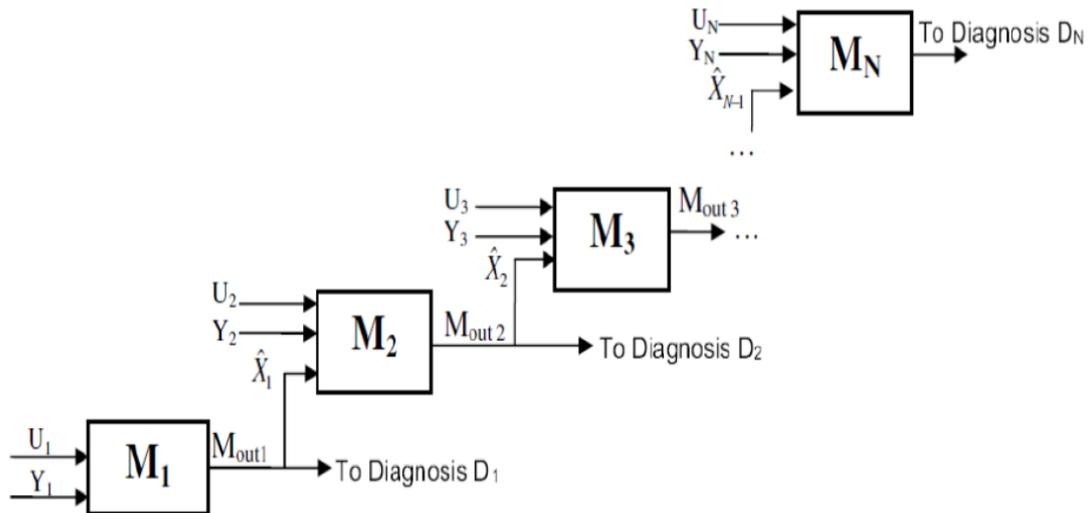


Abbildung 3.4.2: Zustandsraummodell einer hierarchischen Überwachung ([Mek10])

Kompatibel mit dieser Zerlegung ist die RNBC-Struktur. Im Gegensatz zu einer offenen Architektur, in die jede Ebene auf jede beliebige tiefere Ebene zugreifen kann ([Rum91]), handelt es sich bei der RNBC-Struktur um eine geschlossene Architektur; d. h. jede Verhaltensebene kann nur auf die unmittelbar darunterliegende Ebene, über fest definierte Schnittstellen, zugreifen. Diese Ebenen sind rekursiv aufgebaut, so dass Iterationen lediglich zwischen der i -ten und $i+1$ -ten Ebene erfolgen können. Jede Ebene in der RNBC-Struktur übernimmt eine spezielle Aufgabe und erzeugt somit ein bestimmtes Verhalten. Eine Sensorfusion ist nicht notwendig, da Sensoren dort, wo sie am frühesten brauchbar sind, benutzt werden. Abbildung 3.4.3 zeigt eine generische Darstellung der RNBC-Struktur ([Bad94]), die in [Mek09b] und [Luo09] um die Überwachung und Rekonfiguration erweitert wurde. Diese Struktur liefert aufgrund des hierarchischen Aufbaus eine natürliche Möglichkeit, verschiedene Fehlerdetektionen und Rekonfigurationstechniken in einer vereinheitlichten Architektur zu integrieren. Dieses Konzept sieht vor, dass jede Ebene durch einen „Monitor“ überwacht wird, um eventuelle Fehler im System zu erkennen und zu diagnostizieren. Die Überwachung (Monitoring) und Rekonfiguration gehören zusammen zu einem Zwei-Stufen-Konzept für die Verbesserung der Systemverlässlichkeit.

die neue Situation anpasst bzw. die Notwendigkeit an Gegenmaßnahmen (Rekonfiguration) signalisiert. Somit ist die Stabilität des Systems eine Voraussetzung für eine robuste Überwachung.

- Einheitliche verteilte Modellierung trotz verschiedener Abstraktionsebenen: Die unterschiedlichen Verhaltensebenen können unterschiedliche Systembeschreibungen, gemäß der Abstraktionsebene, besitzen. Durch den Einsatz eines Überwachungsverfahrens, das sowohl die kontinuierliche bzw. diskrete Modellierung als auch die hybride Modellierung unterstützt, erhält man eine verteilte Überwachung mit einheitlichen Modellschnittstellen. Diese Eigenschaft erfüllt die Kombination der Differentialgleichungen (kontinuierliche Modelle) mit den Petri-Netzen (diskrete Modelle) in einer neuen Form von hybriden Petri-Netzen, die im nächsten Abschnitt eingeführt werden.

In Kapitel 6 wird eine mögliche Implementierung der Struktur bei mobilen Robotern vorgestellt, für die eine Überwachung bestimmter Verhaltensweisen in dieser Arbeit entwickelt wird.

4 Einbettung der Überwachung in die Systementwicklung

Um eine systematische Entwicklung der Überwachung zu ermöglichen, wird eine Koordination zwischen den verschiedenen Disziplinen des Entwicklungs- bzw. Validierungsprozesses benötigt. Dies hat als Ziel, Sicherheitsaspekte und Überwachungsmechanismen in einer frühen Phase der Systementwicklung einzubetten und damit möglichst effizient - in Bezug auf Zeit und Kosten - das Gesamtsystem zu überwachen. Weiterhin soll die Integration der Überwachung in Teststrukturen dazu beitragen, dass die Verlässlichkeit des Systems erhöht wird. Das gesamte Konzept für die Einbettung der Überwachung in Kobra, unter Berücksichtigung der klassischen Entwicklungsschritte, wird in Abschnitt 4.1 präsentiert. Die Entwicklung eines verteilten verhaltensbasierten Überwachungsverfahrens erfordert eine Systemstruktur, die ein gewünschtes komplexes Verhalten aus mehreren elementaren Verhalten zusammensetzt. Eine solch generische Systemstruktur wird in Abschnitt 4.2, in Zusammenhang mit der in dieser Arbeit entwickelten Überwachung, vorgestellt. Auf beide Phasen der Entwicklung der Überwachungskomponente nach Kobra, nämlich die Entwurfs- bzw. die Realisierungsphase, sowie auf die Integration der Überwachung in Teststrukturen und die Verlässlichkeitsanalyse wird in den Abschnitten 4.3 bzw. 4.4 und 4.5 eingegangen. Zum Schluss werden die verschiedenen Konzepte anhand von Beispielen in Abschnitt 4.6 veranschaulicht.

4.1 Das Konzept der systematischen Entwicklung einer Überwachung

Der in dieser Arbeit entwickelte, modellangetriebene Ansatz für die Überwachung soll die Anforderungen der Systementwicklung auf den verschiedenen Abstraktionsebenen berücksichtigen. Somit wird die Überwachungskomponente, ähnlich wie alle anderen Systemkomponenten, realisiert und während des Betriebs eingesetzt, um das Verhalten des Systems entlang einer Testtrajektorie bzw. einer Mission auf Verlässlichkeit zu überwachen. Abbildung 4.1.1 zeigt das Integrationskonzept der Überwachung im Entwicklungsprozess, in Zusammenhang mit den MPPN.

Mit dem Start der Entwicklung eines technischen Systems ist eine Reihe von Anforderungen zu beachten, die analysiert werden müssen. Das Ziel dieser Anforderungsanalyse ist, eine Spezifikation zu entwickeln, die dann als Grundlage für den Entwicklungsprozess dient. Wichtige Aspekte einer Anforderungsanalyse für sicherheitskritische Systeme, die einen Zusammenhang mit dieser Arbeit darstellen, sind:

- Die funktionalen Anforderungen, also die Menge der Funktionen, die das System erfüllen muss.
- Einige Systemanforderungen, wie Sicherheit, Performanz und Verlässlichkeit. Wobei letztere die vorherigen beinhaltet und das Vertrauen beschreibt, das dem System entgegengebracht wird.

Mit Hilfe der oben genannten Aspekte wird die Anforderungsspezifikation entwickelt. Diese legt die Beschreibung der Struktur und der Schnittstellen sowie die Beschreibung der einzelnen Komponenten mit den jeweiligen nicht funktionalen Anforderungen fest. Somit entspricht die Anforderungsspezifikation der höchsten Entwurfsebene des Systems.

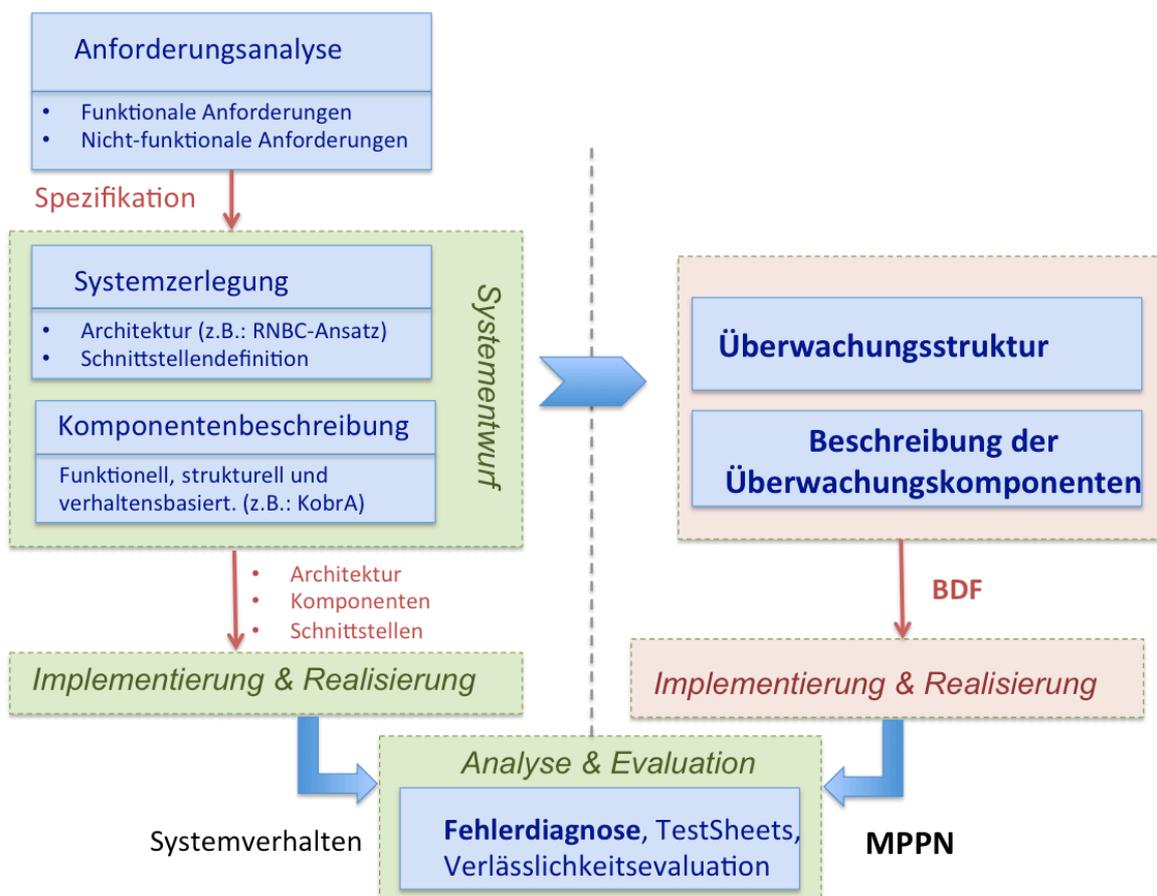


Abbildung 4.1.1: Integration der Überwachung in der Entwicklung und Evaluation dynamischer Systeme

In der Entwurfsphase wird das technische System in Teilsysteme bzw. Komponenten zerlegt, um deren Komplexität gering zu halten. Diese *Zerlegung* ergibt eine Systemarchitektur, die für die Überwachung entscheidend ist. Denn ein gutes Überwachungsverfahren soll die Fähigkeit besitzen, bei verteilten Systemen auf verschiedenen Abstraktionsebenen für verschiedene Modellierungsformen zu operieren, ohne jedes Mal eine neue Beschreibungsform auszusuchen oder neue Modellierungselemente zu definieren. Eine solche

Architektur, an deren unterschiedlichen Abstraktionsebenen eine hierarchische Überwachung in dieser Arbeit angewendet wird, ist die *RNBC-Struktur* (engl.: „Recursive Nested Behaviour-based Control Structure“) ([Bad91]). Es handelt sich um eine rekursive verschachtelte verhaltensbasierte Steuerungsstruktur, die das Gesamtsystem in einzelne Verhaltensebenen zerlegt. Diese werden im Rahmen dieser Arbeit als Verhaltenskomponenten in Kobra beschrieben. Gleichzeitig wird die Überwachungskomponente, in Zusammenhang mit Kobra und den MPPN, in beiden Phasen der Systementwicklung beschrieben (Abbildung 4.1.2). Hierfür wird eine generische Vorlage „BDF“ (engl., „Behavioural description File“) entwickelt, die die Modellsemantik im Kontext der Spezifikation und Realisierung unterstützt und dem Systementwickler als systematische Methode für die Überwachungsentwicklung zur Verfügung steht.

Spezifikation

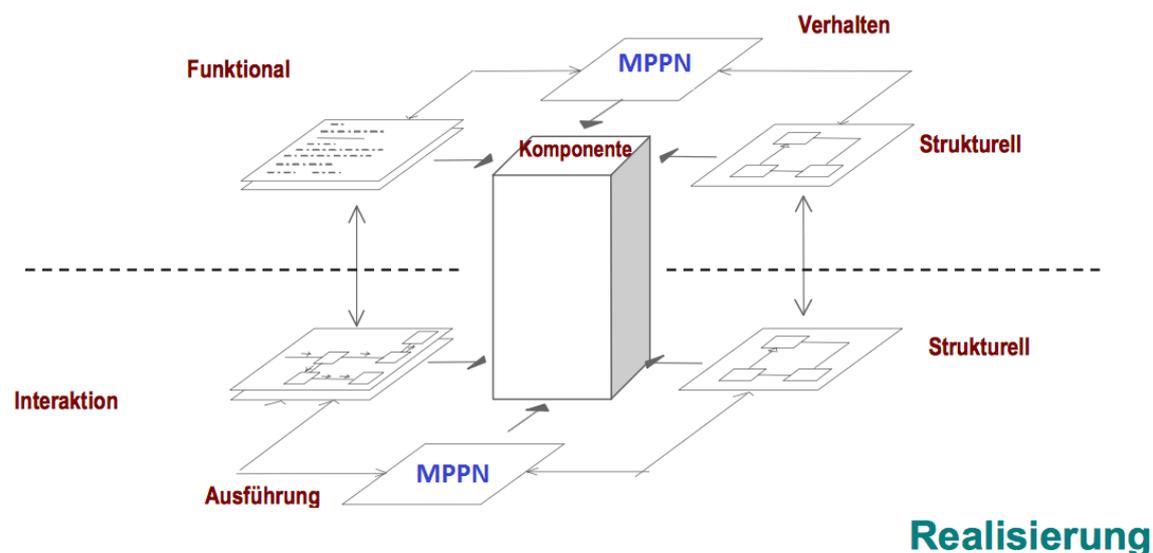


Abbildung 4.1.2: MPPN und Kobra in der Spezifikations- und Realisierungsphase

Mit *Built-in-Test* wird in der Entwurfsphase off-line sichergestellt, ob das System unter Standardbedingungen fehlerfrei läuft. Allerdings herrschen durch den Einsatz des Systems in einer realen Umgebung und aufgrund der Interaktion in dem Moment andere Randbedingungen. Dies macht die on-line Überwachung zu einem notwendigen Prozess in der Teststruktur, sowohl für die Analyse (Fehlerdiagnose) als auch für die Evaluation der Sicherheitsattribute (Verlässlichkeit).

4.2 Verhaltensüberwachung innerhalb verschachtelten Strukturen

Das Hauptmerkmal des verhaltensbasierten Systementwurfes ist die Zerlegung eines komplexen Systems in kleinere handhabbare Subsysteme. Die komponentenbasierte Methode Kobra favorisiert die verschachtelte Komponentenzерlegung, um diese Komplexität gering zu halten. Durch diese Zerlegung ergibt sich ein geordneter Satz von Ebenen, welcher mit dem Konzept der hierarchischen Überwachung in Kapitel 3 kompatibel ist. Innerhalb einer Ebene werden Komponenten, die verwandte Funktionalitäten F_i bereitstellen (funktionale Zerlegung) oder zusammen ein bestimmtes Verhalten B_i realisieren (verhaltensbasierte Zerlegung), zusammengruppiert (Abbildung 4.2.1). Diese Verhaltensebenen (B_1, B_2, \dots, B_N) interagieren miteinander über definierte Schnittstellen, um ein bestimmtes (gemeinsames) Ziel zu erreichen.

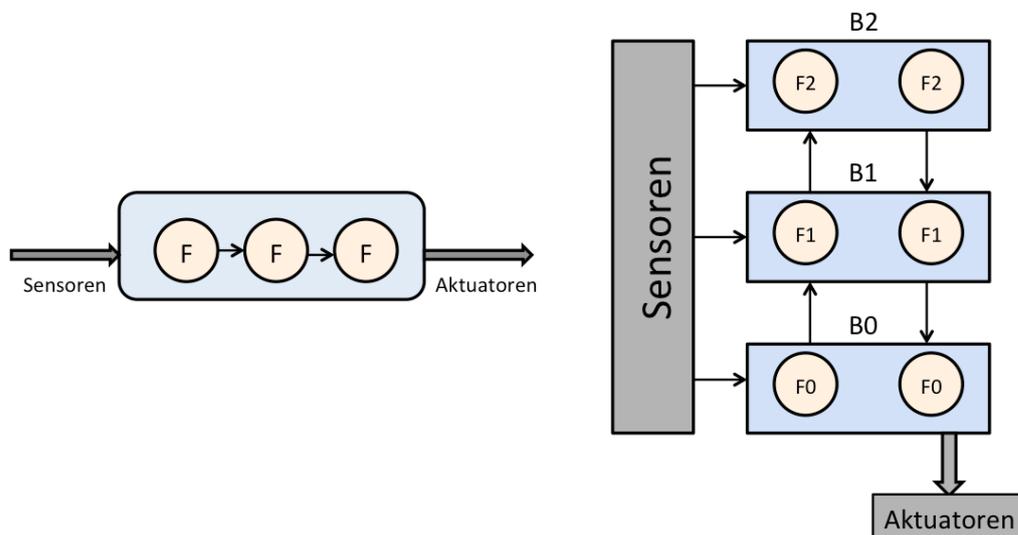


Abbildung 4.2.1: Funktionale und verhaltensbasierte Zerlegung

Um ein dynamisches System zu überwachen, im Sinne eines Vergleichs zwischen dem erwarteten Verhalten und dem tatsächlichen Verhalten, müssen wir zunächst das Verhalten des Systems und seine Interaktionen in einer hinreichenden Abstraktion modellieren können. Hierfür eignet sich der verhaltensbasierte Formalismus von Willems (Abschnitt 2.2.1). In Anlehnung an die Verhaltensdefinition von dynamischen Systemen ist Σ_s die Zustandsraumdarstellung von Σ :

$$\Sigma_s = (T, W, X, B_s). \quad (4.2.1)$$

Für das Verhalten B von Σ gilt $B = \{w \mid \exists x \text{ so dass } (w, x) \in B_s\}$. Die Systemvariablen bilden einen Vektor $w^T = [w_r^T \ w_m^T \ w_i^T]$. w_m ist die Menge der messbaren Variablen, w_r ist die Menge der unmessbaren Variablen, die für die Schätzung relevant sind und w_i ist die Menge

der unmessbaren und für die Schätzung irrelevanten Systemvariablen. Im Fall eines hybriden Systems mit kontinuierlichen bzw. diskreten Ein- und Ausgängen ist:

$$w = u_D \cup u_C \cup y_D \cup y_C \quad (4.2.2)$$

Durch die Anwendung von Willems Definitionen zu gekoppelten Systemen ([Wil97]) werden die Verhaltensebenen der Struktur in Abbildung 4.2.2 als gekoppelte dynamische Systeme definiert. Es gilt:

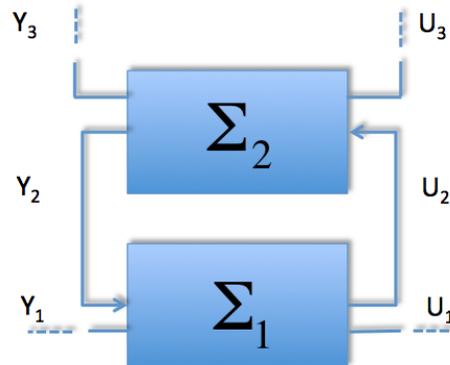


Abbildung 4.2.2: Die Verhaltensebenen als gekoppelte dynamische Systeme

Eine Verhaltensebene Σ_1 , die mit einer darüber liegenden Ebene Σ_2 kommuniziert, stellt das dynamische System $\Sigma_1 = (T, W_1 \times W_2, B_1)$ dar, und es gilt $\Sigma_2 = (T, W_2 \times W_3, B_2)$. Somit ist das resultierende Verhalten aus dieser Kopplung:

$$B = \{(w_1, w_2, w_3): T \rightarrow W_1 \times W_2 \times W_3 \mid (w_1, w_2) \in B_1 \text{ und } (w_2, w_3) \in B_2\} \quad (4.2.3)$$

Im Gegensatz zu Arbeiten zur Modellierung und zum Beobachterentwurf ([Bis05]-[Val99]-[Wil89]), im Rahmen des verhaltensbasierten Formalismus, wird hier im Signalraum W zwischen Eingangssignalen U und Ausgangssignalen Y unterschieden. Somit ist $W = X \times Y$. Die Modellierung der Zustandsübergänge erfolgt durch die Zustandsübergangsfunktion $f: D_f \rightarrow X$ mit $D_f = X \times U$. Diese Abbildung lässt sich folgendermaßen definieren :

$$D_f \subset \{(t_1, t_2, x, u) \mid t_1, t_2 \in T, t_1 \leq t_2, x \in X, u \in U^{[t_1, t_2]}\} \quad (4.2.4)$$

T ist die Menge der Zeitpunkte aus \mathbb{R}^+ . Die Zustandsübergangsfunktion f gibt, abhängig vom aktuellen Zustand, einen Folgezustand an, wenn die entsprechenden Ereignisse aufgetreten sind. Für die Ausgangsabbildung gilt: $g: D_g \rightarrow Y$ mit $D_g = T \times X$.

Diskrete Modelle in Zusammenhang mit dem verhaltensbasierten Formalismus wurden bisher ([Wil89]-[Moo99]) als Zustandsmaschine realisiert. In dieser Arbeit werden Petri-Netze als

diskrete Verhaltensrealisierung eingesetzt ([Mek11c]), und der kontinuierliche Teil des Systemverhaltens B bzw. B_s wird durch Differentiale bzw. Differenzgleichungen realisiert. Somit wird die Spezifikation des externen Verhaltens von Σ_s mit den folgenden $B_{s,Spec}$ beschrieben:

$$B_{s,Spec} := \{w \in W^N \mid \exists x \in X: \forall k \in \mathbb{N}: (x(k), w(k), x(k+1)) \in \xi \text{ und } x(0) \in X'_0\}, \quad (4.2.5)$$

$W, X, X'_0 \subseteq X$, $\xi \subseteq X \times W \times X$ sind der externe Signalraum bzw. der hybride Systemzustand, die Anfangsbedingungen und die Zustandsübergangsbeziehung. Geht man davon aus, dass das Petri-Netz $PN_s = (P, \Gamma, Pre, Post, M_0)$ die diskrete Realisierung von $B_{s,Spec}$ ist, wird für jeden Zustand $x \in X$ eine Stelle $p(x)$ erzeugt. Es ergibt sich eine Menge P an Stellen:

$$P = \{p(x): x \in X\} \quad (4.2.6)$$

Wird PN_s durch ein MPPN ersetzt, wird P in numerische Stellen P_N und symbolische Stellen P_s aufgeteilt. Für jedes Ereignis $e \in \xi$, das zu einem Zustandswechsel von x auf x' führt, wird eine Transition $\tau(e)$ definiert, wobei $\Gamma = \{\tau(e): e \in \xi\}$ die Menge aller Transitionen im MPPN ist. Die Anfangsmarkierung M_0 beschreibt den Anfangszustand von Σ_s .

Pre und $Post$ sind die Vorwärts- bzw. Rückwärtsinzidenzmatrix mit der Dimension $|P| \times |\Gamma|$. Die Inzidenzmatrix $C(p, \tau) = Post(p, \tau) - Pre(p, \tau)$ beschreibt gleichzeitig die Interaktion des numerischen Teils mit dem symbolischen. Solch eine Überwachung, basierend auf einer Zustandsschätzung innerhalb einer Verhaltensstruktur, die aus separaten Verhaltensebenen besteht, wird im nächsten Abschnitt am Beispiel der RNBC-Struktur eingeführt. Die Verhaltensebenen in dieser Struktur kommunizieren über fest definierte Schnittstellen, die wiederum die Schnittstellen zum Überwachungssystem darstellen. Abbildung 4.2.3 veranschaulicht dieses Konzept anhand von zwei ineinander verschachtelten Komponenten bzw. Teilsystemen. Die Verallgemeinerung auf n -Teilsysteme ist analog dazu darstellbar. Das interne Teilsystem S_1 besteht aus einer Strecke „Subsys.1“, einem Regler C_1 und einem eigenen Überwacher M_1 . S_1 ist in dem Teilsystem S_2 verschachtelt, welches wiederum aus „Subsys.2“, C_2 und M_2 besteht. Anhand der Steuereingänge U_1, U_2 bzw. Ausgänge Y_1 und Y_2 ermitteln M_1 und M_2 die Zustände der Teilsysteme. Die Ergebnisse jedes Überwachers können entweder nach außen oder an den Überwacher des nächst höheren Subsystems weitergegeben werden.

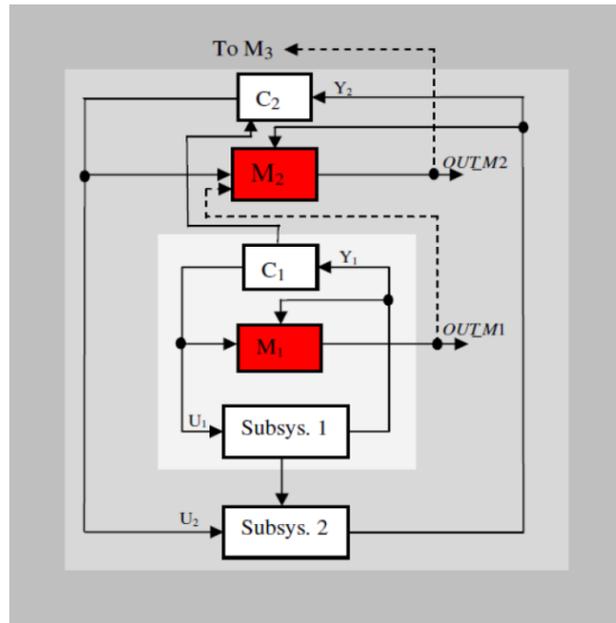


Abbildung 4.2.3: Das Konzept der Verschachtelung ([Mek10])

Beispiel: Zur Veranschaulichung einiger Konzepte in dieser Arbeit, wie z. B. die Verschachtelung der Komponenten und die Beschreibung der Überwachung in Kobra, sollte das Beispiel „Thermostat“ auf eine Heizungsregelung in einem Raum bzw. in einem Haus erweitert werden. Die Abbildung 4.2.4 zeigt ein HTCS „House Temperature Control System“ mit drei Räumen.

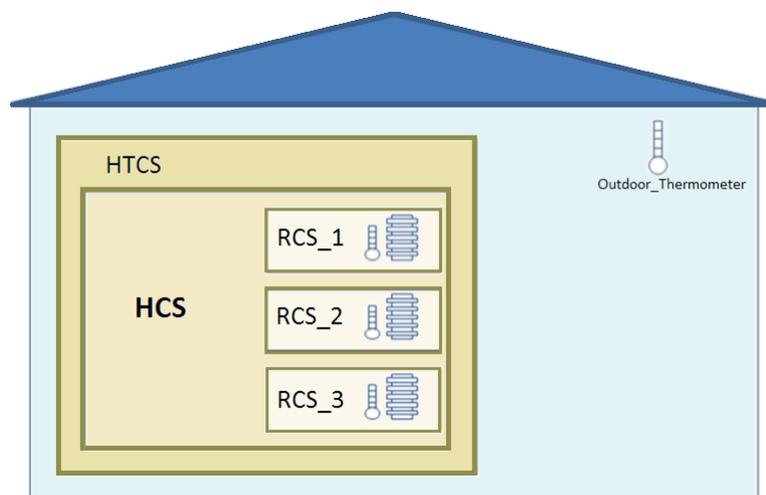


Abbildung 4.2.4: Temperaturregelung in einem Haus

Die Temperatur θ in jedem Raum wird mit Hilfe eines Thermostats geregelt, der kontinuierlich durch einen Sensor die Temperatur im Raum misst und die Heizung je nach Bedarf ein- und ausschaltet (RCS: „Radiator control system“) (Abbildung 4.2.5). Wenn die Heizung ausgeschaltet ist, sinkt die Raumtemperatur, die durch die Variable θ beschrieben ist, nach dem Verlauf der Exponentialgleichung $\theta(t) = \theta_0 e^{-Kt}$. Dabei ist t die Zeit, θ_0 die

Anfangstemperatur und K ist eine vom Raum abhängige Konstante. Ist die Heizung angeschaltet, folgt die Temperatur der Funktion $\theta(t) = \theta_0 e^{-Kt} + h(1 - e^{-Kt})$. In dieser Gleichung beschreibt h eine Konstante, deren Wert von der Heizungsleistung abhängt.

Die Temperaturänderung in einem Haus mit mehreren Räumen lässt sich in der Literatur ([Feh04]) wie folgt beschreiben:

$$\dot{\theta}_i = \alpha_i + \beta_i(\theta_{\text{out}} - \theta_i) + \sum_{j \neq i} \gamma_{i,j}(\theta_j - \theta_i) \quad (4.2.7)$$

$\gamma_{i,j}$, β_i und α_i sind Konstanten, wobei $\gamma_{i,j} = 0$ falls die Räume i und j keine benachbarten Räume sind und α_i beträgt 0, wenn die Heizung im Zimmer ausgeschaltet oder im Ruhezustand ist. Die Temperatur eines Raumes hängt von der Differenz der Temperatur mit den anderen Räumen, von der Differenz mit der Außentemperatur θ_{out} und vom Modus der Heizung, ob ein- oder ausgeschaltet, ab.

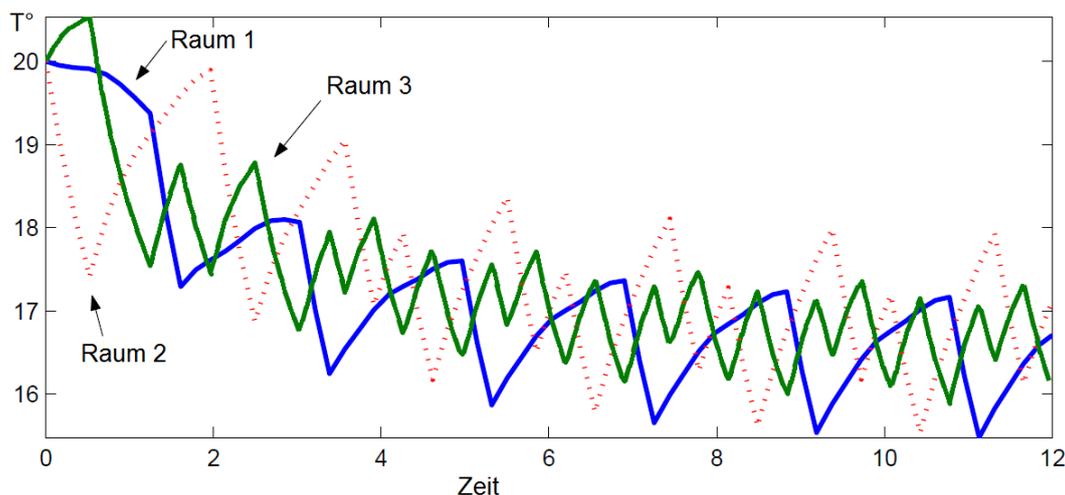


Abbildung 4.2.5: Die Temperaturentwicklung in den Räumen

Eine verschachtelte Zerlegung dieses Beispiels lässt sich durch das Blockdiagramm in Abbildung 4.2.6 darstellen. Es ergeben sich drei Ebenen: die Ebene der graphischen Benutzerschnittstelle „HUMAN-COMMAND“, das Temperaturregelungssystem des Hauses „HTCS“ und die Ebene der Raumtemperaturregelung „RCS“, hauptsächlich aus einem „Thermostat“ mit einem Heizkörper-Regelsystem „Radiatoren“ bestehend. Durch die Bildung des Mittelwerts der Temperaturen in allen Räumen wird aus der Haustemperaturregelung eine Raumtemperaturregelung und aus dem HTCS ein „Heater Control System – HCS“. Dieses HCS ist für die Aufrechterhaltung der Temperatur eines angegebenen Wertes verantwortlich. Der Prototyp in Abbildung 4.2.7 simuliert ein Demo für das HCS-Beispiel als Raum „Room“, bestehend aus Raumtemperatursensoren, einem Außentemperatursensor, zwei Lüftern auf der Vorderseite (als Störquellen) und ein Heizkörper-Regelsystem „Thermostat“, das wiederum

aus einem Heizkörper „Heater“, Temperatursensor und Heizkörperregler „Radiator“ besteht. Die gewünschten Temperaturwerte werden durch den Nutzer über eine graphische Benutzerschnittstelle gegeben.

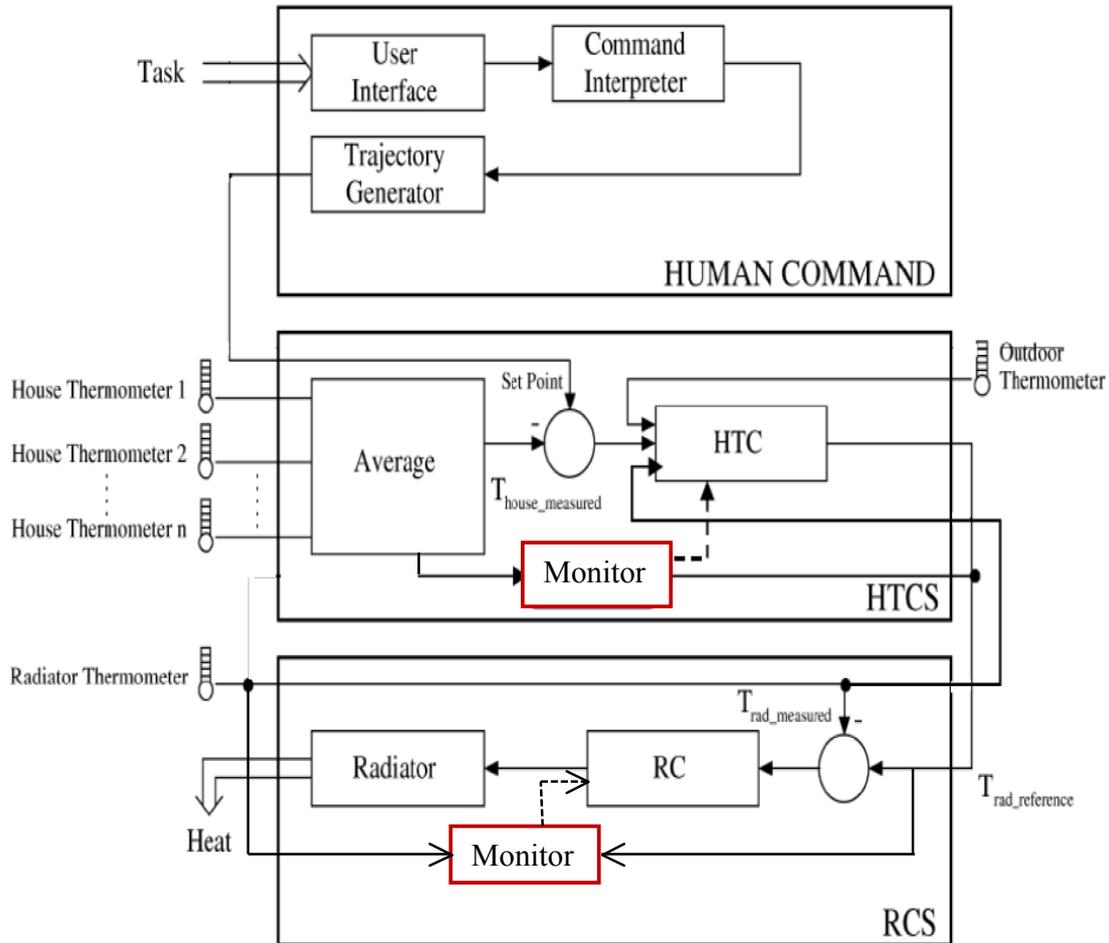


Abbildung 4.2.6: Hierarchische Zerlegung des HTCS (JMEK10)

Die Dynamik des Heizkörpers in Abbildung 4.2.6 wird durch die folgende Gleichung beschrieben:

$$\dot{\theta}_{\text{rad}} = \alpha_1 + \beta_1(\theta_{\text{room}} - \theta_{\text{rad}}) \quad (4.2.8)$$

θ_{room} und θ_{rad} sind die Temperaturen des Raums bzw. des Heizkörpers. Die Temperaturänderung des Heizkörpers $\dot{\theta}_{\text{rad}}$ hängt von der Differenz beider Temperaturen ab. Die Dynamik des Teilsystems „HCS“ ergibt sich aus der Temperaturänderung im Raum $\dot{\theta}_{\text{room}}$. Diese hängt sowohl von der Temperatur des Heizkörpers als auch von der Differenz mit der Außentemperatur ab:

$$\dot{\theta}_{\text{room}} = \alpha'_1 + \beta'_1(\theta_{\text{rad}} - \theta_{\text{room}}) + \beta_2(\theta_{\text{out}} - \theta_{\text{room}}) \quad (4.2.9)$$

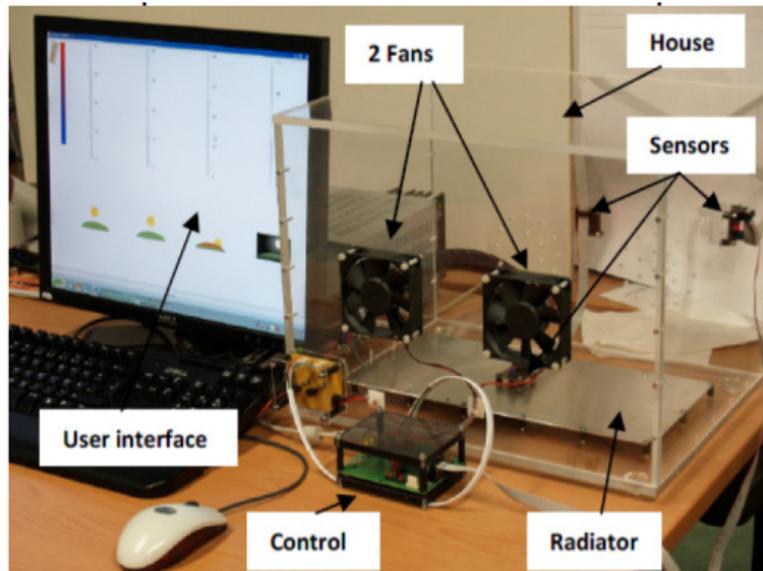


Abbildung 4.2.7: Demonstrations-Plattform zum HCS ([Mek10])

Die Abbildung 4.2.8 gibt einen Überblick über die Komponenten des HCS und über ihre zur Verfügung gestellten und verwendeten Schnittstellen. Es zeigt sehr gut die verschachtelte Hierarchie des Systems und dass die Komponenten nur über klar definierte Schnittstellen interagieren.

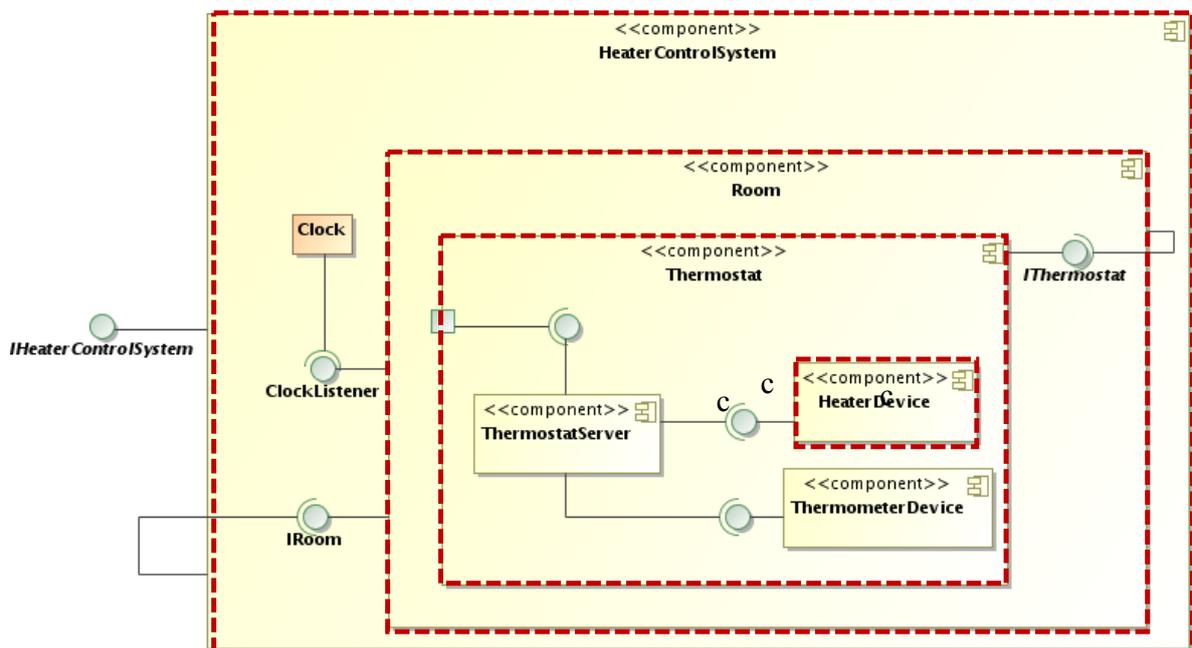


Abbildung 4.2.8: HCS-Komponentenübersicht

Nach dem Konzept der hierarchischen Überwachung besitzt jede Ebene einen Überwacher, der sich auf der nächsthöheren Ebene befindet. Somit liegt der Überwacher vom Heizkörper „Heater“ auf der Ebene „Thermostat“ und enthält ein hybrides Modell, dessen Zustandswechsel nur von der kontinuierlichen Dynamik, also von der Entwicklung der Temperatur, abhängt. Liegt diese unter einem vorgegebenen Schwellwert soll die Heizplatte

heizen („Heating“). Ist die obere Temperaturschwelle überschritten wird es nicht mehr geheizt („Cooling“). Der Überwacher vom Raum „Room“ liegt auf der Ebene „HCS“ und enthält sowohl die Informationen über die Entwicklung der Temperatur als auch über die zwei Betriebsmodi „ON“ und „OFF“ des Thermostats. Schaltet der Nutzer den Thermostat ein („switch On“) findet der Zustandswechsel nach der Regeldynamik der unteren Ebene statt, also je nach Temperaturwert wird es geheizt oder gekühlt. Schaltet der Nutzer das System aus („switch Off“), wechselt das System unabhängig von der vorliegenden Temperaturdynamik zum Zustand „OFF“. Abbildung 4.2.9 zeigt die verschachtelte Verhaltensbeschreibung der Ebenen. Die Beschreibung der beiden Überwacher nach Kobra ist in Abbildung 4.2.10 dargestellt.

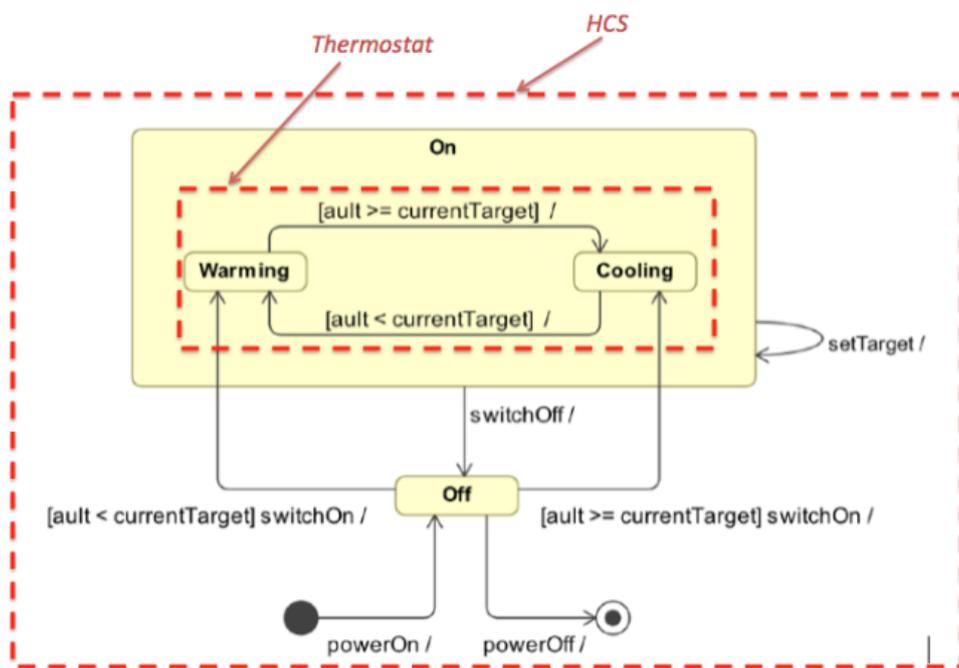


Abbildung 4.2.9: Die verschachtelten Überwachungsebenen im HCS

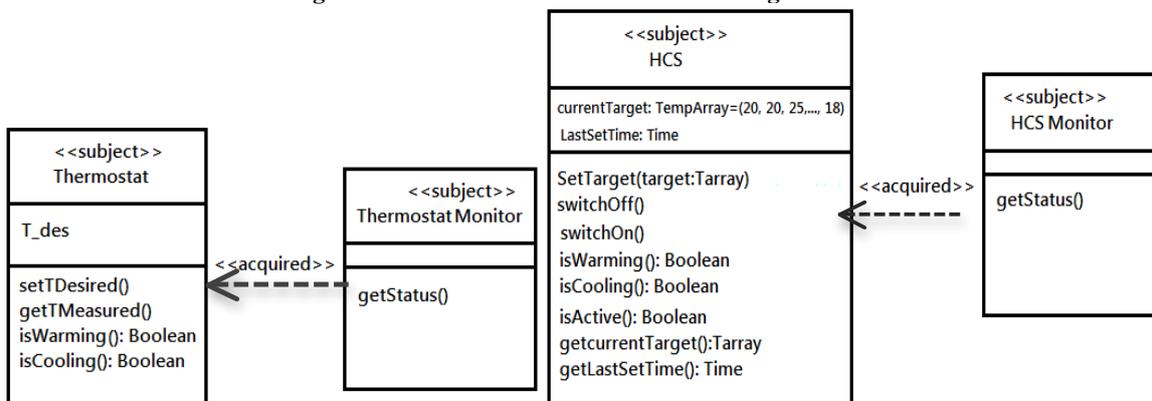


Abbildung 4.2.10: Beschreibung der Überwachung im HCS mittels Klassendiagrammen

4.3 Formaler Überwachungsentwurf in Kobra

Voraussetzung zur Anwendung des modellbasierten Ansatzes für die Überwachung ist, dass das zu überwachende System durch ein ausführbares Verhaltensmodell repräsentiert wird. Der Einsatz von Petri-Netzen in dieser Arbeit, sowohl für die formale Beschreibung des Systemverhaltens als auch für die Fehlerdiagnose, hat eine nahtlose Integration der Überwachung in allen Phasen des Entwicklungsprozesses ermöglicht. Dies bietet einerseits den Vorteil von werkzeugunterstützter Verifikation, und andererseits können die durch MPPN entwickelten Modelle für die Realisierung der Überwachung wieder verwendet werden. Die Abbildungen 4.3.1 und 4.3.2 zeigen die ausführbaren Verhaltensmodelle der Komponenten „Thermostat“ und „Heater“ als Petri-Netze unter Kobra.

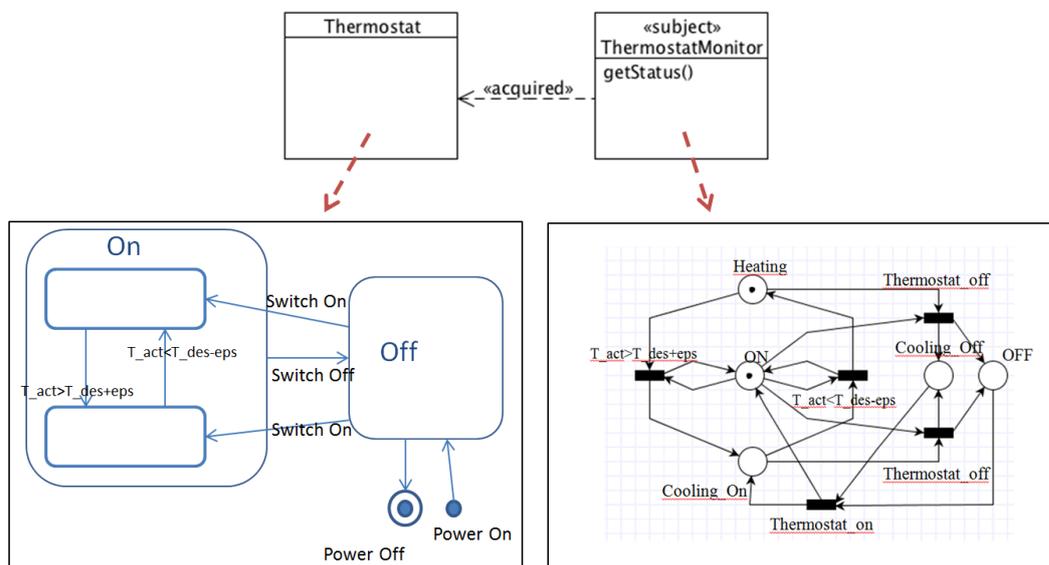


Abbildung 4.3.1: Formale Beschreibung des Thermostat-Überwachers

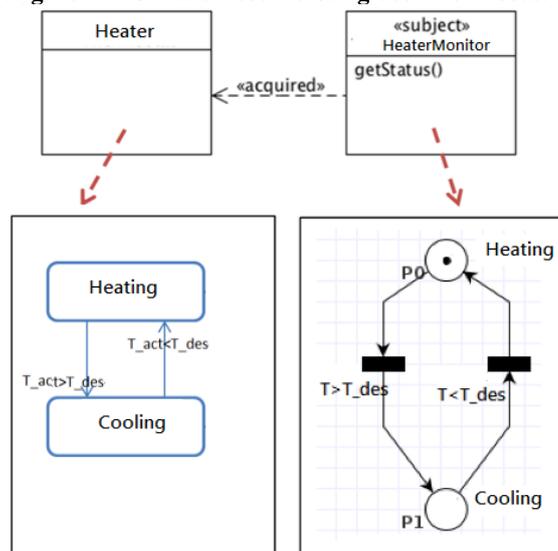


Abbildung 4.3.2: Formale Beschreibung des Heizkörper-Überwachers

Die Beschreibung der generischen Überwachungskomponente unter Kobra besteht, wie bei allen anderen Komponenten, aus den verschiedenen Sichten. Die Abbildung 4.3.3 und 4.3.4 zeigen die Spezifikation und die Realisierung des MPPN-basierten Überwachers (Monitor) einer zu überwachenden Komponente CUM (Component Under Monitoring).

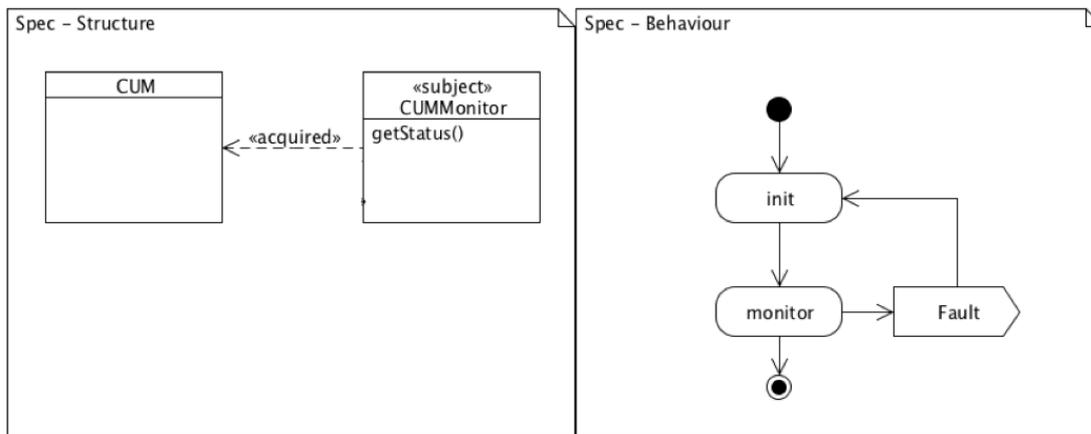


Abbildung 4.3.3: Spezifikation der Überwachungskomponente ([Wag11])

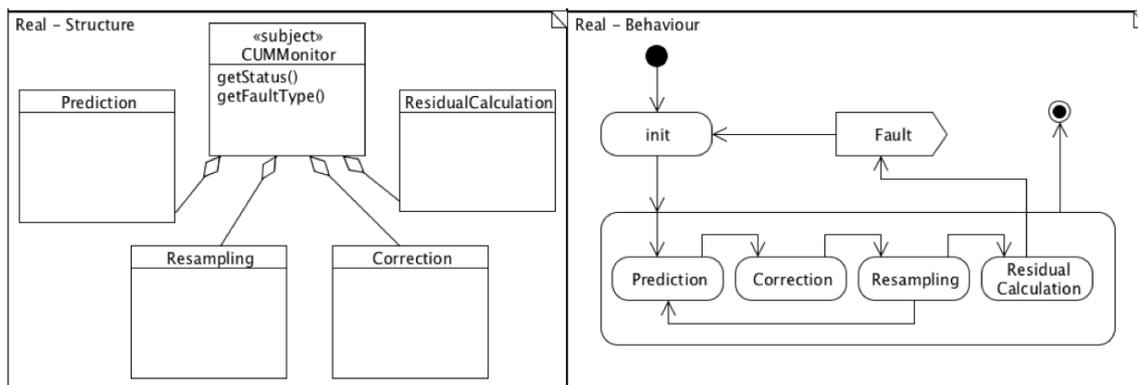


Abbildung 4.3.4: Realisierung der Überwachungskomponente ([Wag11])

Das Problem der Überwachung in der Entwurfsphase ist wie folgt formuliert: Basierend auf Willems Definition für die Zustandsraumdarstellung $\Sigma_s = (T, W, X, B_s)$ eines dynamischen Systems Σ und seine Verhaltensspezifikation $Spec$ ist ein Überwacher zu entwerfen. Dieser soll den hybriden Systemzustand X unter Berücksichtigung der Unsicherheit schätzen, um festzustellen, ob das System Σ_s die Spezifikation $Spec$ erfüllt. Der Lösungsansatz hierfür besteht aus den folgenden Schritten:

- Entwurf einer generischen Spezifikationsvorlage $Spec' \subseteq Spec$, die nur Elemente aus $Spec$ enthält, welche als wesentlich für die Überwachung gehalten werden. Diese Elemente sind z. B. die zu überwachenden Variablen, eine Verhaltensspezifikation $B_{S,Spec}$ von Σ_s als die Menge der akzeptablen Signale $B_{S,Spec} \subseteq W^T$, die Rauschbedingungen des Systemrauschens bzw. der Systemgenauigkeit sowie die

Anzahl der Partikel und deren Initialwerte, die a priori definiert werden können oder mit einem Zufallsgenerator erzeugt werden etc.

- Das Einbetten von $Spec'$ im Überwacher Σ_{monit} , um das Systemverhalten mit seiner Spezifikation zu vergleichen.

In dieser Arbeit wird das Überwachungssystem Σ_{monit} ebenso als ein dynamisches System $\Sigma_{monit} = (T, W, X, B_{monit})$ definiert, dessen Prozesse nebenläufig mit den Prozessen des Zielsystems $\Sigma_s = (T, W, X, B_s)$ ausgeführt werden. Die Aufgabe von Σ_{monit} ist eine Schätzung des hybriden Systemzustands \hat{X} zu liefern und, basierend darauf, festzustellen, ob Σ_s während des Betriebs seine Spezifikation $Spec$ erfüllt. Die Abbildung 4.3.5 zeigt die Architektur des Ansatzes und die Schnittstellen zum zu überwachenden System. Die Vorlage BDF zur Verhaltensbeschreibung von Σ_{monit} enthält sowohl die notwendigen Teilspezifikationen $Spec'$ aus der Gesamtspezifikation $Spec$ als auch die notwendigen Parameter für die Initialisierung des Überwachers (Abbildung 4.3.6). Diese lassen sich aus den folgenden Elementen zusammenstellen:

- **Definition der Systemparameter**

Unter "*Netparameters*" werden die Elemente des zu überwachenden numerischen Zustandsvektors x_N definiert. Sie befinden sich an den numerischen Stellen, deren Entwicklung die kontinuierliche Dynamik des Systems abbildet (z. B. Temperatur, Geschwindigkeit, Position). Die "*Configurations*" stellen die Elemente des zu überwachenden symbolischen (diskreten) Zustandsvektors x_S dar. Diese befinden sich an den symbolischen Stellen, deren Entwicklung vom Auslösen der Ereignisse abhängt.

- **Definition der Sicherheitsparameter**

Es handelt sich um die maximale zulässige Abweichung ε „*Max Deviation*“, die sich auf die kontinuierliche Dynamik des Systems bezieht und Bedingungen über den Zustandswechsel im kontinuierlichen Teil des Modells festlegt. Diese Abweichungen werden aus der Systemspezifikation abgeleitet.

- **Definition der Systemparameter**

- Für die numerischen Stellen "*NUM*" werden die Koeffizienten der zugeordneten Differenzgleichungen, die das kontinuierliche Systemverhalten modellieren, aufgelistet (z. B. die Koeffizienten A und B einer Differenzgleichung der Form

$x_{k+1} = A \cdot x_k + B \cdot u_k$). [P0][NUM][0.9971][0.9986] zeigt die Stelle P0 mit der Gleichung: $x_{k+1} = 0,9971 \cdot x_k + 0,9986 \cdot u$.

- Für die symbolischen Stellen "SYM" werden die zugeordneten Konfigurationen aufgelistet. Diese stellen z. B. eine Systemkonfiguration „OP_MODE“ (Betriebsmodus) bzw. eine Umgebungskonfiguration (Navigationsbereich eines Fahrzeugs) dar.

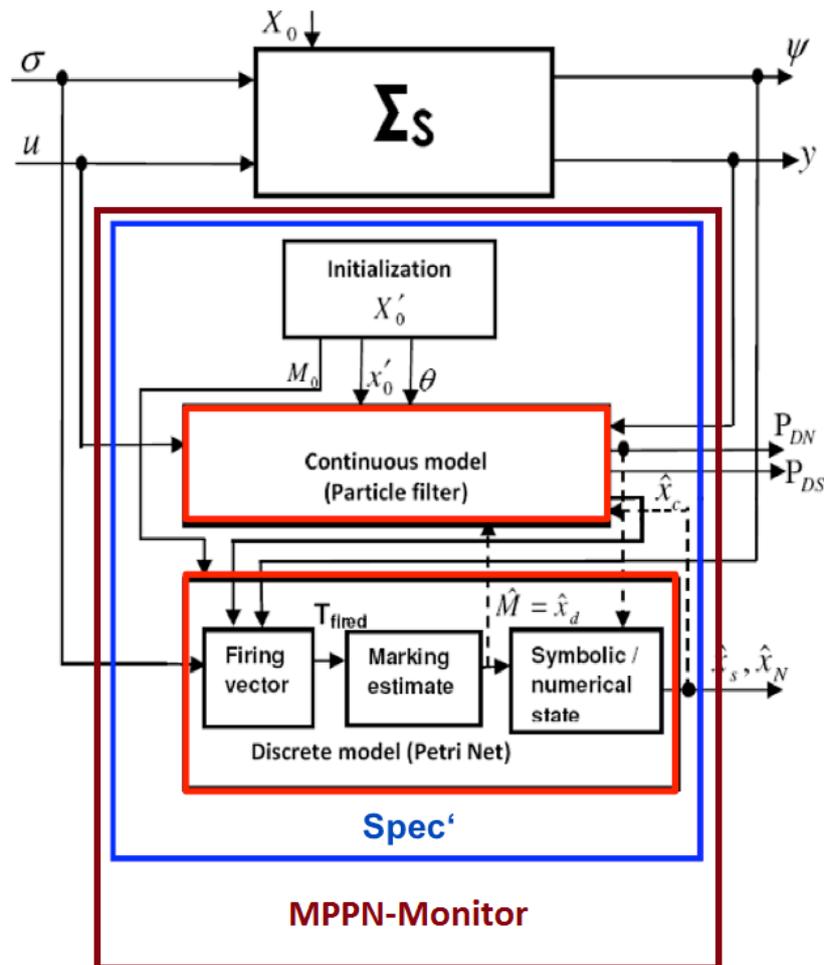


Abbildung 4.3.5: Struktur des hybriden Überwachers und Schnittstellen zum System

- **Definition der Bedingungen für Systemübergänge**

Auslöser für die numerischen Übergänge ist das Überschreiten von kontinuierlichen Schwellwerten an den numerischen Transitionen „Numerical_Trans“. Diese Eingangsbedingungen können einfache Bedingungen sein, welche Vergleichsoperatoren verwenden, einschließlich "<", ">", ">=", "<=" oder "=" oder auch diese in Kombination mit den logischen Operatoren "AND" oder "ODER". Weitere Arten von Bedingungen sind Intervalle $[I^-, I^+]$, so genannte "Intervall Conditions". An den symbolischen Transitionen

werden die Bedingungen für einen Moduswechsel „*MODE*“ aufgrund eines externen Ereignisses definiert.

```

>Netparameters number_of_param ←  $X_N$ 
[param_1] [param_2] ...
>Configurations number_of_config ←  $X_S$ 
[config_1] [config_2] ...
>Max_Deviation number_of_dev_param ←  $\varepsilon$ 
[Dev_param][dev_param=dev_value]
>Places number_of_places
[PO][NUM][A11 A12][B1]
[P1][NUM][A21 A22][B2] }  $B_{S,Spec}$ 
...
[OP_MODE_1][SYM][ ]
[OP_MODE_2][SYM][ ]
...
>Transitions number_of_transitions ←  $u$ 
[numerical_Trans1][NUM][T][param_1>value_a]
[numerical_Trans2][NUM][A][param_2>value_b & param_2<value_c]
[numerical_Trans3][NUM][O][param_3>value_d || param_3<value_e]
...
[OP_MODE_1][SYM][MODE = 0.0]
[OP_MODE_2][SYM][MODE = 1.0] }  $\sigma$ 
...
>Incidence matrix ←  $C$ 
-1 1 0 -1 0
1 -1 -1 0 1
...
>Pred_noise [value_p_noise] ←  $\eta$ 
>Obs_noise [value_n_noise] ←  $v$ 
>Marking number_of_particles
[PO][NUM][init_value]
[P0][NUM][init_value]
...
[OP_MODE_1][SYM][1.0]

```

Diagramm zur generischen Beschreibung der Überwacher-Struktur. Die Textuelle Vorlage ist in drei Hauptbereiche unterteilt:

- Spec'** (rot markiert): Umfasst die Parameter X_N , X_S , ε , $B_{S,Spec}$, u und σ .
- Initialization X_0'** (rot markiert): Umfasst die Parameter η , v und die Marking-Parameter.
- Σ_{monit}** (blau markiert): Umfasst die Parameter C und die OP_MODE-Parameter.

Abbildung 4.3.6: Textuelle Vorlage zur generischen Beschreibung der Überwacher-Struktur.

- **Definition der Modellstruktur**

Die Modellstruktur wird durch die Inzidenzmatrix festgelegt. Diese beschreibt die Verknüpfung zwischen Stellen und Transitionen bzw. den Zusammenhang zwischen den Zuständen und der aktiven Entwicklung der Markierung.

- **Definition der Unsicherheiten / Systemrauschen/ Modellierung von Störungen**

Die beobachtungsbedingten Unsicherheiten aus dem Messmodell werden als Messrauschen bezeichnet. Die Unsicherheit in den Zuständen wird durch das Prozessrauschen modelliert.

- **Definition der Anfangsbedingungen**

Der Anfangszustand des Systems wird durch eine Anfangsmarkierung, die die Anzahl der Tokens (Partikel und Konfigurationen) und deren Initialwerte enthält, beschrieben.

4.4 Überwachungsrealisierung durch Wiederverwendung von Komponentenbeschreibungen

Grundgedanke: durch die Wiederverwendung von Komponentenbeschreibungen im komponenten-basierten Entwicklungsprozess wird die verhaltensbasierte Realisierung einer funktionalen Komponente als Beobachtermodell für die Überwacher-Spezifikation verwendet und für den MPPN-basierten Überwachungsprozess eingesetzt. Abbildung 4.4.1 stellt das Konzept der Wiederverwendung von Komponentenbeschreibungen dar.

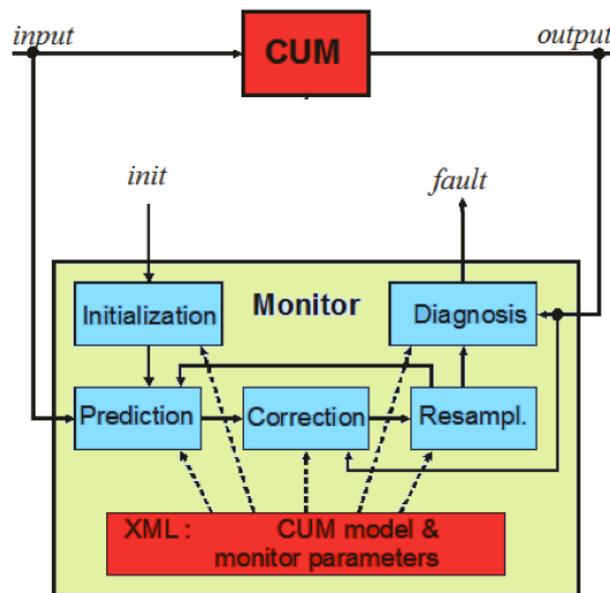


Abbildung 4.4.1: XML-Vorlage im Überwachungsprozess ([Wag11])

Sei „CUM, component under Monitoring“ eine funktionale Komponente, die überwacht werden soll. Das Verhaltensmodell für die Realisierung dieser Komponente als Petri-Netz stellt gleichzeitig einen Teil der Überwachungskomponente dar, nämlich das Observer Modell. Nach der automatischen Generierung der Modellspezifikation wird dieses durch die Parameter für die Systeminitialisierung bzw. durch stochastische Attribute für den Schätzungsprozess ergänzt. Man erhält somit die verhaltensbasierte Beschreibung „BDF“ der Überwachungskomponente. Diese wird während des Überwachungsprozesses für die Schätzung des Systemzustands (Prädiktion, Korrektur und Resampling) und für die Fehlererkennung durch den Diagnose-block eingesetzt. Die Abbildung 4.4.2 zeigt den Zusammenhang zwischen dem Überwachungsmodell einer Komponente „CUM“ und die daraus abgeleitete XML-Beschreibung am Beispiel des HCS:

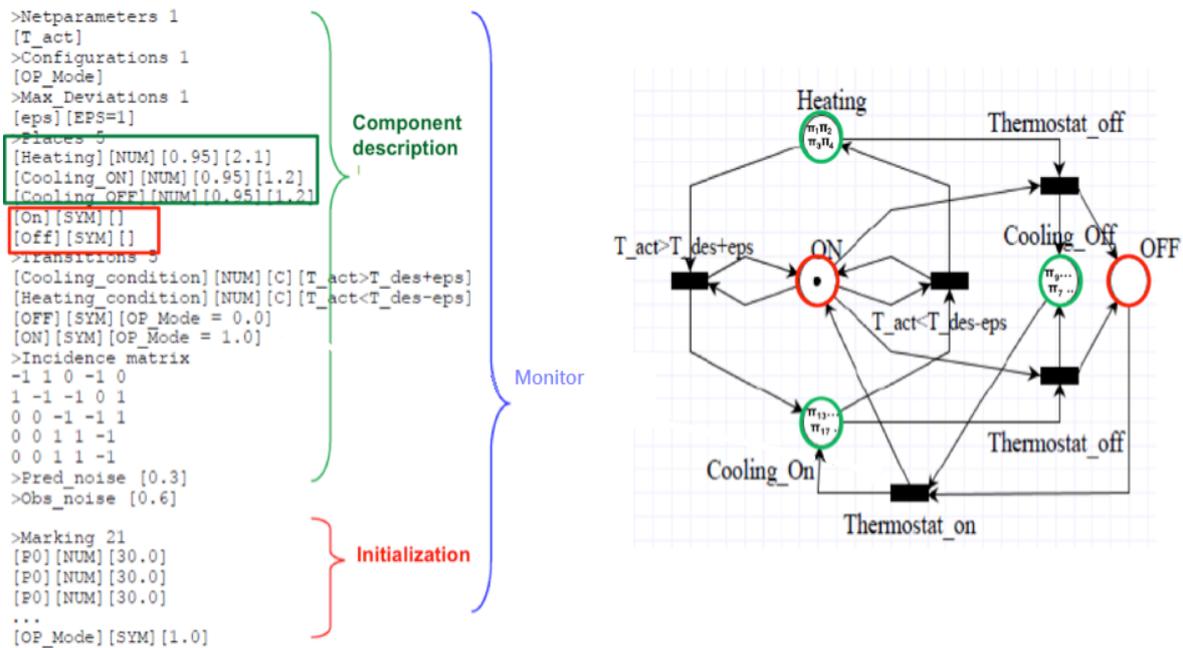


Abbildung 4.4.2: Generierung des XML-BDF aus dem MPPN-Modell des Überwachers

4.5 Überwachung in Teststrukturen und Verlässlichkeitsanalyse

Ein realisiertes System soll während des Betriebs überwacht werden. Die Ergebnisse dieser Überwachung können für weitere Prozesse nützlich sein, z. B. für Tests und Verlässlichkeitsevaluationen. Die Idee in diesem Abschnitt ist die *Integration der Überwachung in Teststrukturen*, um das Systemverhalten zu überwachen und Aussagen über die *Verlässlichkeit* des Systems zu treffen. Dies erfolgt durch einen Vergleich des erwarteten Verhaltens (aus der Spezifikation) mit dem tatsächlichen Verhalten. Diese Abweichungen werden verwendet, um die Attribute der Verlässlichkeit nach [Wag13] zu ermitteln und damit ein Maß für die Verlässlichkeit zu berechnen. Für Tests werden hier die in Kobra entwickelten TestSheets (Ergebnisblatt) verwendet. Diese verwenden, wie bei der Überwachung, die formalen Verhaltensspezifikationen des Systems bzw. Teilsystems und tauschen die Daten mit der Überwachungskomponente über einen Daten-Provider aus. Der Daten-Provider liest die Daten der Test-Mission während der Ausführung des Tests und leitet sie an die Überwachungskomponente weiter (Abbildung 4.5.1). Die Ergebnisse werden im TestSheet gespeichert, und die Verlässlichkeitsberechnung wird durchgeführt und ausgegeben.



Abbildung 4.5.1: Datenaustausch mit dem Test-Sheet für Verlässlichkeit

Abbildung 4.5.2 zeigt das Prinzip der Monitoring-Teststruktur. Der TestSheet enthält die Mission und die Attribute für die Berechnung der Verlässlichkeit ([Wag13]). Die Missionsparameter können direkt aus der Verhaltensspezifikation des Systems wiederverwendet werden. Je nach Systembeschreibung können die Testeingänge entweder als Trajektorien fest definiert werden oder von einem Test-Trajektoriengenerator erzeugt werden (siehe „multi scenarion TestSheets [Atk03]). Während des Systemtests wird die Mission dem Überwachungssystem übergeben. Dieses generiert Eingangsabstastwerte u_k in Echtzeit, die vom Referenzmodell verwendet werden, um die Solltrajektorie zu erzeugen. Der Eingang u_k wird gleichzeitig der zu testenden Komponente und dem Schätzblock überführt. Das Ergebnis dieser Blöcke sind Ausgangssignale, die für die Residuumberechnung zwischen Soll- und Ist-Zuständen und für die weitere Systemdiagnose verwendet werden. Interne Zustände einer Komponente, die nicht messbar sind, werden vom Schätzblock geschätzt und für die Diagnose weiterverwendet.

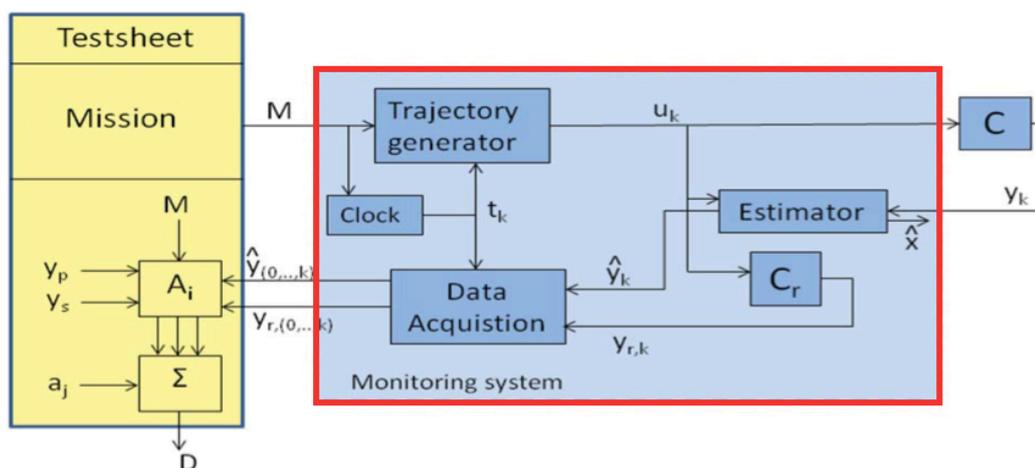


Abbildung 4.5.2: Konzept der Integration in Teststrukturen ([Wag13])

Das Konzept zur Integration der Überwachung in Teststrukturen und die Verlässlichkeitsberechnung werden im folgenden Abschnitt durch Beispiele veranschaulicht.

4.6 Beispiele

Beispiel 1: Temperaturüberwachung im Haus

Im Beispiel des Temperaturregelungssystems nach Abbildung 4.6.1 definiert der Nutzer „Inhabitant“ eine Liste von gewünschten Temperaturwerten für den Tagesverlauf, z. B. $T_{des} = \{30.0, 31.0, 30.5, 31.5, 32.5, 32.5, 33.0, 33.5, 34.0, 33.5, 34.0, 34.5, 33.5, 33.0, 32.5, 32.0, 31.5, 31.5, 31.0, 30.5, 30.5, 30.5, 30.5, 30.5\}$. Der Thermostat regelt und hält die gewünschte Temperatur durch das Wechseln zwischen Heizen und Kühlen.

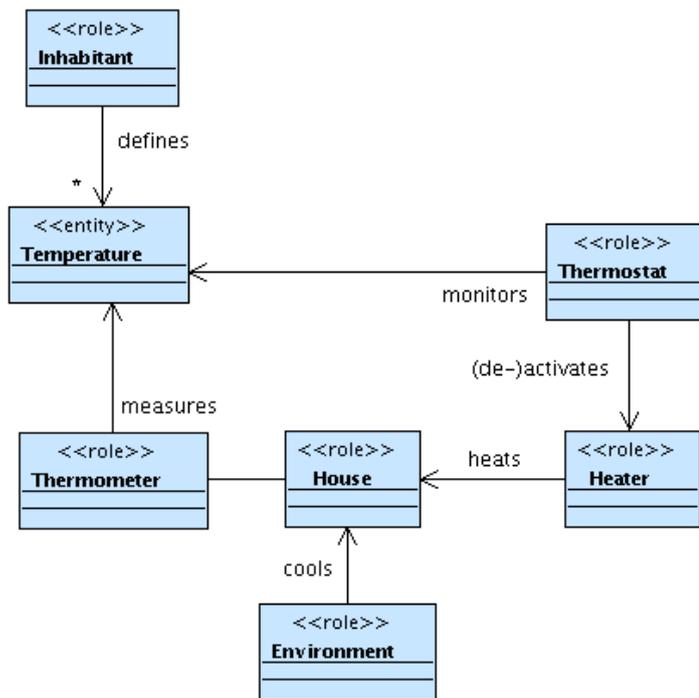


Abbildung 4.6.1: Die Komponenten des HTCS

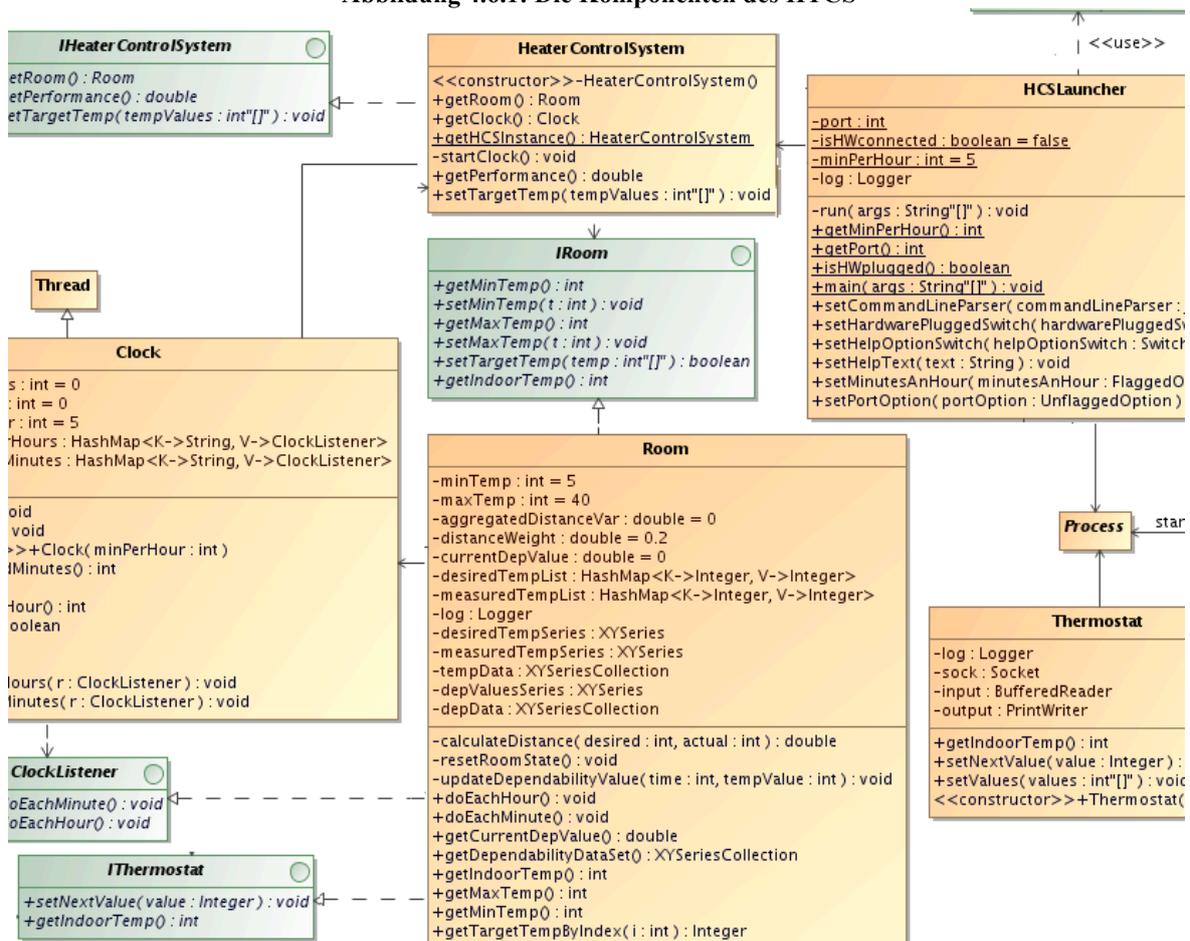


Abbildung 4.6.2: Übersicht der Schnittstellen und Komponentenklassen

Je nach Raumtemperaturentwicklung und unter Einhaltung der Randbedingungen eines sicheren Betriebs, welche aus der Spezifikation (Abbildung 4.6.2) entnommen werden (z. B. ein Betriebstemperaturbereich von $[5^{\circ}\text{C}-40^{\circ}\text{C}]$), bestimmt der Überwacher den wahrscheinlichsten Zustand des Thermostaten (Abbildung 4.6.3).

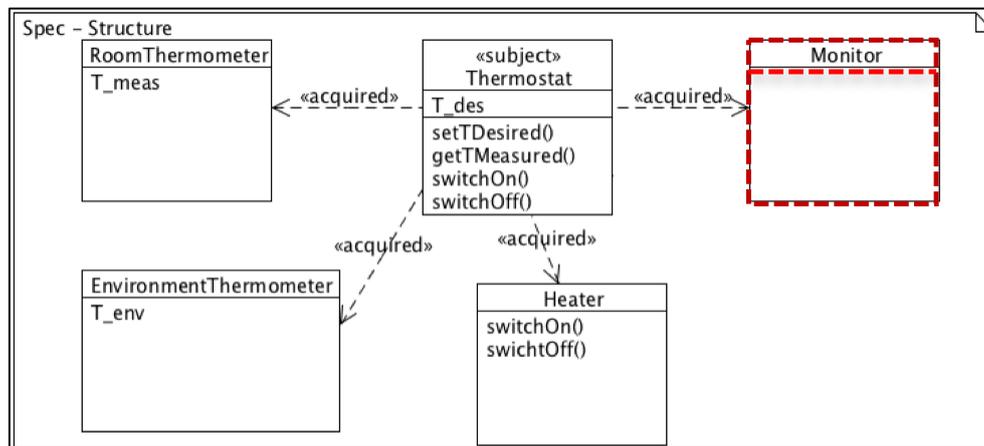


Abbildung 4.6.3: Übersicht der Schnittstellen und Komponentenklassen

Die experimentellen Ergebnisse in Abbildung 4.6.4 zeigen zu jedem Zeitpunkt die vom Nutzer gewünschten Temperaturwerte (Verlauf in rosa Farbe) und den geschätzten höchstwahrscheinlichen Systemzustand (in %-Werten auf der linken Achse). Um einen höheren gewünschten Temperaturwert zu erreichen, soll die Heizplatte heizen, und der höchstwahrscheinliche Systemzustand ist „Heating“. Ist der nächste gewünschte Temperaturwert kleiner, befindet sich das System im „Cooling“ Zustand, und die Temperatur sinkt nun mit einer gewissen, sich leicht ändernden Rate, bis die untere Schwelle erreicht wird.

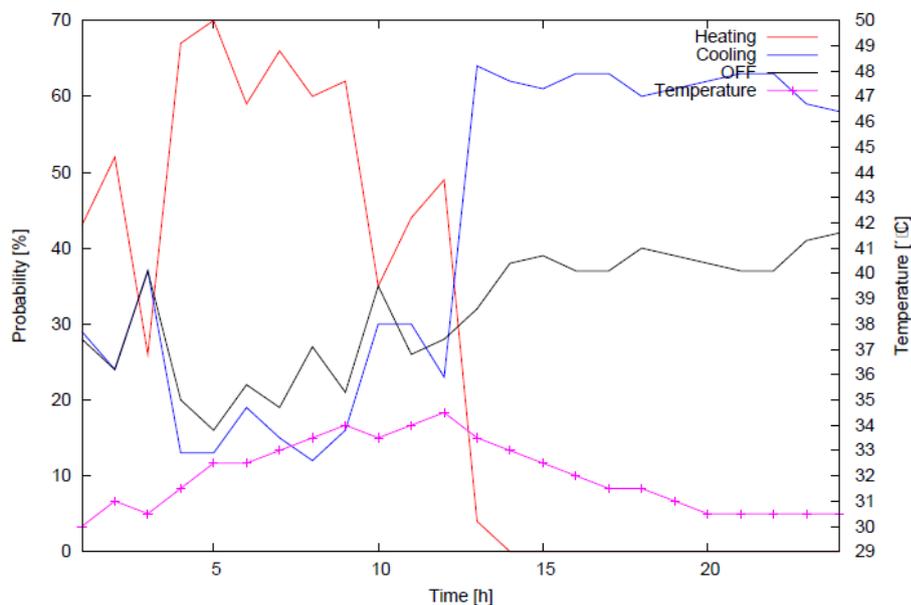


Abbildung 4.6.4: Schätzung der Betriebsmodi

Wird das System durch den Nutzer ausgeschaltet (der Teil aus dem schwarzen Verlauf nach der zweiten Stunde) wird der Zustand „OFF“ als höchstwahrscheinlich geschätzt. Abbildung 4.6.5 zeigt den Testsheet vom HCS, der die gewünschte Soll-Temperatur setzt und das System durch die Definition der unteren und oberen Temperaturgrenzwerte simuliert. Die Korrektheit des Verhaltens wird durch die Abfrage des Zustands bestätigt.

	A	B	C	D
1	Room	create		
2	@D1(first)	setMaxTemp	[10..45 <random>]	
3	@D1(first)	setMinTemp	[0..10 <random>]	
4	@D1(first)	getMaxTemp		C2
5	@D1(first)	getMinTemp		C3
6	@D1(first)	setTargetTemperature	{30.0, 31.0, 30.5, 31.5, 32.5, 32.5, 33.0, 33.5, 34.0, 33.5, 34.0, 34.5, 33.5, 33.0, 32.5, 32.0, 31.5, 31.5, 31.0, 30.5, 30.5, 30.5, 30.5}	
7	@D1(first)	getIndoorTemp		<C2 and >C3
8	sequence	^1	^9	^10
9	inAnyOrder	^2	^3	^6
10	inAnyOrder	^4	^5	^7

Abbildung 4.6.5: TestSheet für die Temperaturentwicklung im Raum

Beispiel 2: Navigation eines autonomen Rollstuhls

Gemäß den Nutzeranforderungen aus der Spezifikation hat ein elektrisch angetriebener Rollstuhl die Aufgabe, in einer strukturierten, teilweise bekannten Umgebung, autonom zu navigieren. Die Zielposition „End“ soll aus einer Startposition „Start“ und über Wegpunkte „WP“, bei einer Vermeidung von Kollisionen, erreicht werden (Abbildung 4.6.6).

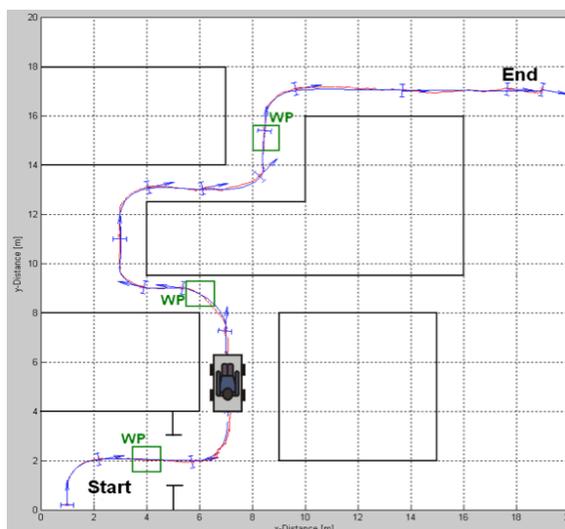


Abbildung 4.6.6: Navigation eines Rollstuhls

Zur Demonstration der hybriden Überwachung mit den MPPN wird die Orientierungsänderung des Rollstuhls überwacht. Im einfachsten Fall wird dieses Rollstuhlverhalten mit drei Zuständen angegeben, nämlich "geradeaus fahren", "links abbiegen" und "rechts abbiegen" (wir betrachten nur die Vorwärtsbewegung). Abbildung

4.6.7 stellt den Monitor des Rollstuhlverhaltens als MPPN und die daraus automatisch generierte textuelle Verhaltensbeschreibung BDF dar.

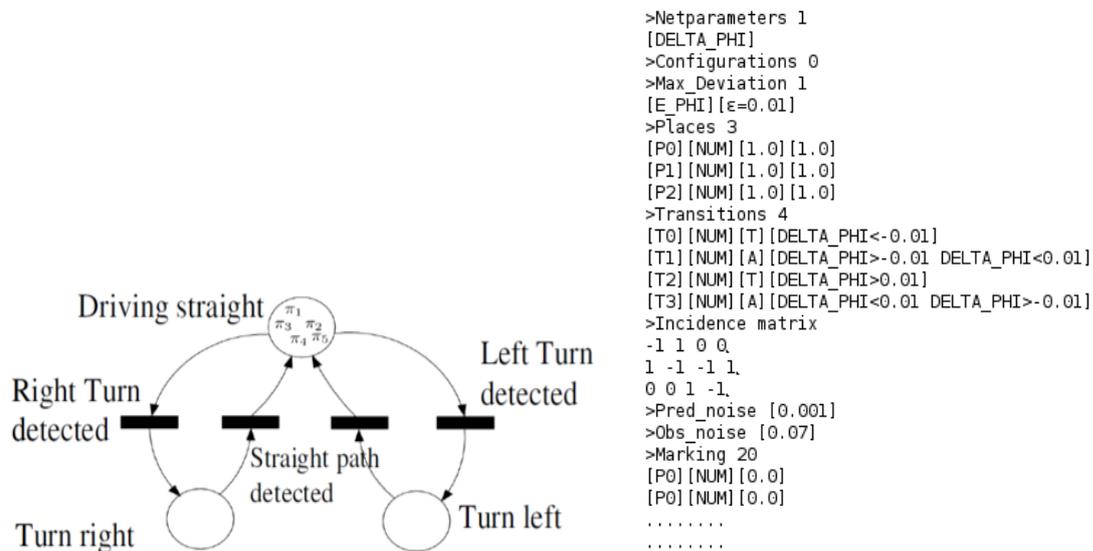


Abbildung 4.6.7: Der Überwacher für die Orientierungsänderung als MPPN und BDF ([Mek11c])

Die Ergebnisse dieser Schätzung sind in Abbildung 4.6.8 dargestellt: Die roten Punkte stellen die Beobachtung jeder Sekunde dar, und die blauen Kreuze sind die Partikel, die die Schätzung der Orientierungsänderung repräsentieren. Die negativen Werte bedeuten, dass eine Rechtskurve und die positiven eine Linkskurve gefahren wird. Die Wahrscheinlichkeiten der Zustände sind auf der rechten Achse eingetragen.

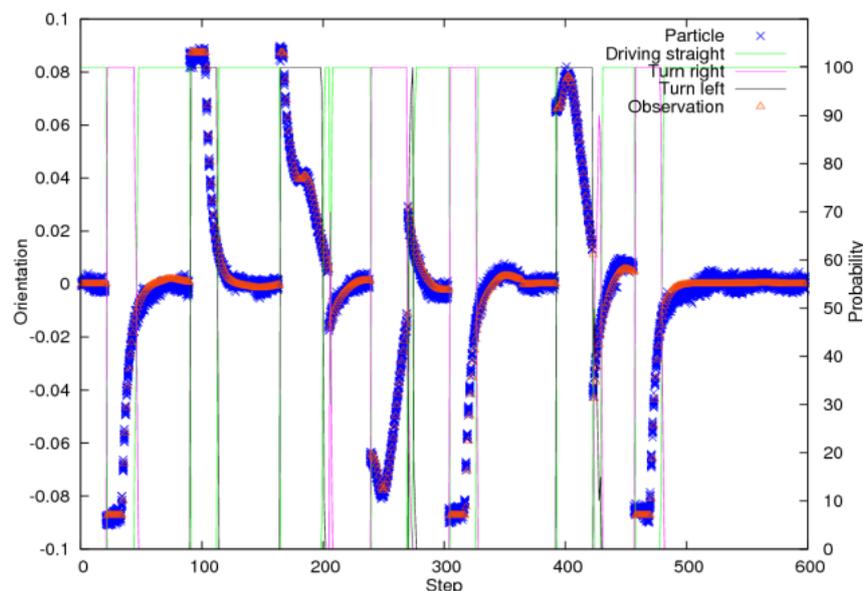


Abbildung 4.6.8: Schätzungsergebnisse im fehlerfreien Fall

Abbildung 4.6.9 zeigt den Fall, dass die Orientierungsabweichung des Geschwindigkeitsreglers größer ist als die spezifizierte maximal erlaubte Abweichung. Der Monitor ist zwar immer noch in der Lage, den kontinuierlichen Systemzustand (Orientierung) korrekt zu

schätzen. Der diskrete Zustand wird jedoch mit höherer Wahrscheinlichkeit falsch geschätzt. Während des Überwachungsprozesses speichert der Testsheet die Ergebnisse des Monitors (Position und Orientierung) und stellt sie dem Verlässlichkeitsberechnungsprozess zur Verfügung. Abbildung 4.6.10 zeigt den Testsheet für die Positionsüberwachung des Rollstuhls.

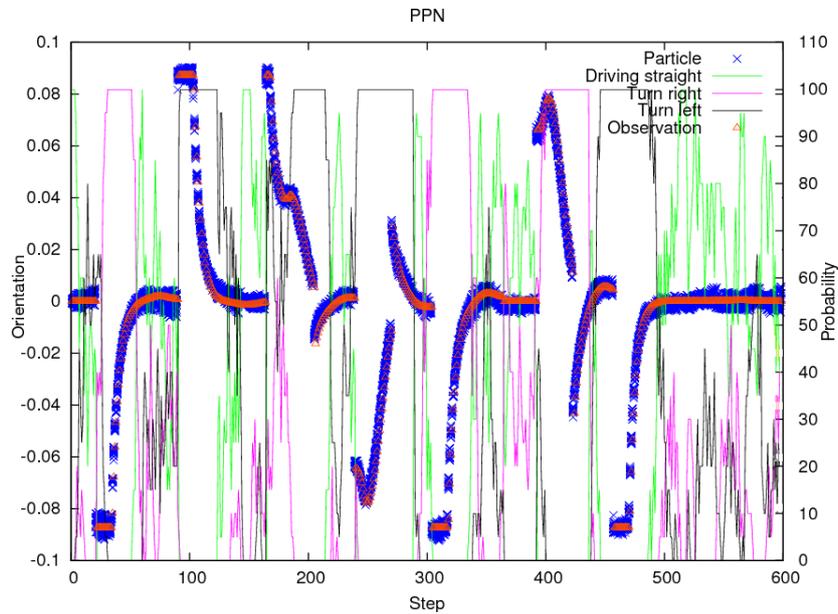


Abbildung 4.6.9: Schätzungsergebnisse im Fall einer Abweichung von der Spezifikation

Die Missionsdaten, die durch den Überwacher geschätzten Positionen sowie die gemessenen Positionen des Rollstuhls werden für die Arbeit [Wag13] zur Verfügung gestellt, um daraus die Verlässlichkeitsattribute „Safety“ und „Performance“ zu ermitteln und für die Verlässlichkeitsvalidierung nach dem Konzept des Verlässlichkeits-TestSheets (Abbildung 4.5.2) einzusetzen.

	A	B	C	D
1	IWCCconnector	create		
2	C1	getNextDataSet		
3	C2	getXMission		
4	C2	getYMission		
5	C2	getXEstimation	D5 < 0.5	[C3-C5]
6	C2	getYEstimation	D6 < 0.5	[C4-C6]
7	100% -> 8 / 1 2			
8	[C2 != null] -> 8 / 3 4 5 6 2	[C2 == null] -> 9		
9				

time= 37.500	x_mission= 4.4145	y_mission= 14.8791	x_est= 4.0485	y_est= 14.5148	phi_mission= 0.6133	vx= 1.0000	vphi= -0.5102
time= 37.600	x_mission= 4.4655	y_mission= 14.9105	x_est= 4.1055	y_est= 14.5885	phi_mission= 0.5517	vx= 1.0000	vphi= -0.5476
time= 37.700	x_mission= 4.5132	y_mission= 14.9360	x_est= 4.1713	y_est= 14.6536	phi_mission= 0.4911	vx= 0.9846	vphi= -0.5588
time= 37.800	x_mission= 4.5578	y_mission= 14.9566	x_est= 4.2159	y_est= 14.7070	phi_mission= 0.4321	vx= 0.8982	vphi= -0.5494
time= 37.900	x_mission= 4.5992	y_mission= 14.9729	x_est= 4.2891	y_est= 14.7676	phi_mission= 0.3750	vx= 0.8371	vphi= -0.5616
time= 38.000	x_mission= 4.6373	y_mission= 14.9855	x_est= 4.3424	y_est= 14.8203	phi_mission= 0.3203	vx= 0.7479	vphi= -0.5646
time= 38.100	x_mission= 4.6723	y_mission= 14.9951	x_est= 4.4005	y_est= 14.8592	phi_mission= 0.2682	vx= 0.6817	vphi= -0.6034
time= 38.200	x_mission= 4.7043	y_mission= 15.0023	x_est= 4.4307	y_est= 14.8723	phi_mission= 0.2190	vx= 0.6158	vphi= -0.5871
time= 38.300	x_mission= 4.7334	y_mission= 15.0073	x_est= 4.4919	y_est= 14.9177	phi_mission= 0.1729	vx= 0.5835	vphi= -0.5059

Abbildung 4.6.10: Versorgung des TestSheets durch Überwachungsdaten.

5 Entwicklungs-, Simulations- und Analyseumgebung

Gegenstand dieses Kapitels ist es, ein Werkzeug zu entwickeln, welches den Entwurf und die on-line Ausführung der in Kapitel 3 entwickelten MPPN ermöglicht. Die Komponenten des entwickelten Werkzeugs werden in Abschnitt 5.1 vorgestellt. Diese stellen die wesentlichen Beiträge zum praktischen Teil der Arbeit dar. In Abschnitt 5.2 wird der MPPN-Netzeditor „EXT-PIPE“ (EXTended Platform Independent Petri net Editor) als Erweiterung eines bereits vorhandenen offencode Petri-Netzeditors eingeführt. Die automatische Generierung der textuellen generischen Vorlage für die Beschreibung des Verhaltensmodells, die in Kapitel 4 im Realisierungsprozess integriert wurde, erfolgt mittels eines Kompilers „Net-Compiler“, der in Abschnitt 5.3 vorgestellt wird. Der Kern der on-line und Echtzeitüberwachung mittels MPPN ist in dieser Arbeit ein in C/C++ entwickelter Simulator „MPPN-Sim“, dessen Implementierung als Pseudo-Code in Abschnitt 5.4 dargestellt wird.

5.1 Komponenten des Überwachungswerkzeugs

Das hier entwickelte Überwachungswerkzeug ermöglicht die komfortable Entwicklung eines Systemmodells mittels Petri-Netzen sowie die zustandsschätzungs-basierte Überwachung der Prozesse mittels Partikelfilter, und zwar in einem einheitlichen Framework. Das Werkzeug integriert zu diesem Zweck einen Eingabeeditor für Petri-Netze, einen Compiler und einen Simulator für MPPN. Das Zusammenspiel der einzelnen Komponenten der Entwicklungsumgebung wird schematisch in Abbildung 5.1.1 dargestellt.

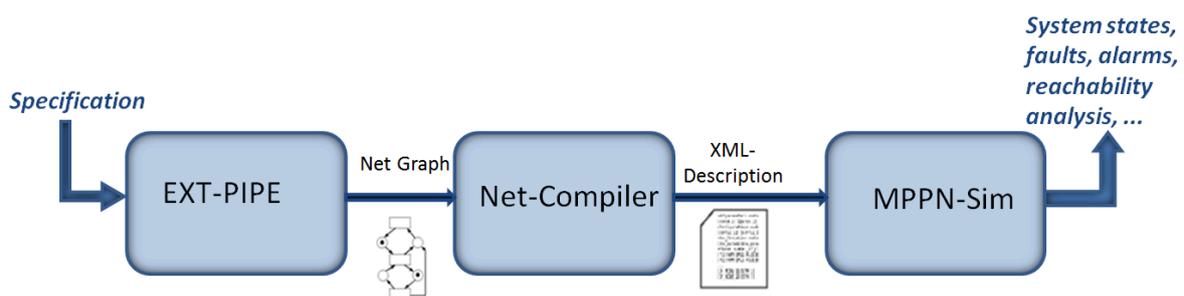


Abbildung 5.1.1: Entwicklungsumgebung der Überwachung

Der Petri-Netzeditor „EXT-PIPE“ dient der graphischen Erstellung des Systemmodells, das aus der Systemspezifikation abgeleitet wird. Nachdem ein Netzmodell eingegeben wird, extrahiert der „Net-Compiler“ aus dieser graphischen Beschreibung sowohl die Netzstruktur als auch systemverhaltensbezogene Elemente, wie z. B. Anfangszustand (Markierung), Verhaltensgleichungen, Sicherheitsparameter, Betriebsmodi und weitere Elemente. Diese

werden in einem zweiten Schritt vom Kompilierer in einem textuellen kompakten XML-Austauschformat (engl., „Extensible Markup Language“) transformiert, welche sich an PNML-Standard (engl., „Petri Net Markup Language“ ([Bil03])) orientiert. Diese Vorlage, eine so genannte „Verhaltensbeschreibungsdatei“ (engl., „Behavioral Description File“, BDF), wird bei der Simulation exportiert. Damit haben wir einen weiteren Vorteil, nämlich die einfache Möglichkeit der Übersetzung in andere XML-Sprachen, wie sie von existierenden Analysewerkzeugen benutzt werden, was eine leichte Anbindung an diese Tools ermöglicht. Der MPPN Simulator übernimmt die Verwaltung, die Koordination der Simulationsdurchläufe und die Ausgabe der Ergebnisdaten. Die Benutzerinteraktion findet während des gesamten Prozesses über den graphischen Editor bei off-line Simulationen oder über eine Konsole bei on-line Simulationen statt. Sämtliche Daten zu einem MPPN-Netz werden lokal zu dem Netz in einem Netzordner gespeichert. Dazu gehören z. B. Dateien für die textuelle Repräsentation des Netzes, die Parameter-Daten, die Schätzungsergebnisse und weitere Diagnose bezogene Informationen. In den folgenden Abschnitten werden der graphische Netzeditor, der Netzcompiler und der Simulator etwas näher betrachtet.

5.2 Auswahl und Erweiterung des Petri-Netzeditors

Die Implementierung eines Petri-Netzwerkzeugs mit Standardfunktionalität, wie ein graphischer Editor, Visualisierung von Ergebnissen und Funktionen zum Parsen einer Datei ist eine Zeit raubende Angelegenheit. Dieser Aufwand ist hier unnötig, da die werkzeugunterstützte graphische Darstellung von Petri-Netzen nicht das Ziel dieser Arbeit ist, sondern die Implementierung der Kernfunktionalitäten des MPPN-Werkzeuges. Hierfür wurde ein bereits vorhandener Petri-Netzeditor verwendet und eigenständig erweitert, anstatt die Basisfunktionalität zu implementieren, welche schon in vielen Petri-Netzeditoren implementiert wurden. Für diese Arbeit wurde PIPE2 [Blo03] in seiner erweiterten Version EXHOST-PIPE ([Bon06a]-[Bon06b]) verwendet, welche vom französischen Luft- und Raumfahrt-Forschungszentrum ONERA um den Entwurf und die Simulation der Partikel Petri-Netze erweitert wurden. PIPE2 zeichnet sich durch die Anwenderfreundlichkeit der Bedienoberfläche und die Möglichkeiten zur selbstständigen Anpassung und Erweiterung durch Schnittstellen und Quellcodeoffenheit aus. Es ist auf zwei Arten erweiterbar: Zum einen durch das Schreiben von Modulen, die über einfache Programmierschnittstellen in das Programm eingebunden werden können, auch ohne Kenntnis des dahinterliegenden Codes. Zum anderen ist das Programm quelloffen und damit frei modifizierbar. Die Erweiterung von PIPE2 auf EXHOST-PIPE war erforderlich, um Partikel Petri-Netze simulieren zu können.

Hierfür wurden Buttons für die Erstellung symbolischer und numerischer Stellen sowie passender Transitionen hinzugefügt. Zusätzlich können letztere mit Modus-Änderungen bzw. Bedingungen versehen werden; diese Einstellungen werden in einer Informationsleiste sichtbar gemacht. Numerische Stellen können Differenzgleichungen übergeben, symbolische Stellen dann Konfigurationen. Weitere Buttons erlauben das Erzeugen von Partikeln sowie Konfigurationen und die Einstellung der Simulationsparameter. Abbildung 5.2.1 stellt die Benutzeroberfläche der EXHOST-PIPE-Software dar. Dieser Petri-Netzeditor und -simulator war jedoch zu großen Teilen auf einen bestimmten Anwendungsfall (Flugplanung) ausgerichtet und dementsprechend programmiert worden. Dadurch konnten viele Einstellungen, die für die Erstellung eigener Netze nötig sind, nicht über die Benutzerschnittstelle gemacht werden; sie waren fest codiert. Zusätzlich verhinderten einige Bugs die Anzeige von Grafiken und das Ausführen eigener Simulationen. Das Tool war in Java geschrieben und damit weder für eine Echtzeitüberwachung noch zur Integration ins Steuerungssystem geeignet.

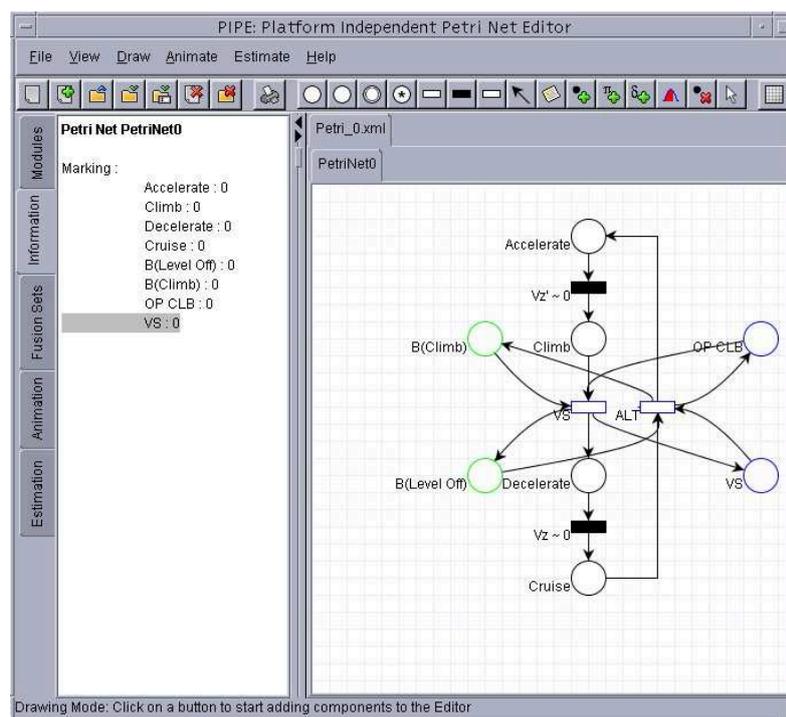


Abbildung 5.2.1: EXHOST-PIPE Benutzeroberfläche

Alle diese Einschränkungen haben dazu geführt, dass in dieser Arbeit ein neues Software-Werkzeug zur Erstellung und Simulation von MPPN entwickelt wurde (Abbildung 5.2.2). Dabei wurde die EXHOST-PIPE graphische Oberfläche (GUI) vom internen Monitoring-Simulator M getrennt. Die GUI wird von EXT-PIPE als Netzeditor und als eine interaktive

Schnittstelle zum Benutzer verwendet. Über EXT-PIPE werden die einzelnen Netzelemente beschriftet, und die MPPN-Modelle werden eingegeben, dargestellt und geändert. Die Markierungen der Stellen, sowohl numerische Markierungen durch Partikel als auch symbolische Markierungen durch Marken bzw. Konfigurationen, werden definiert und initialisiert. Ein Compiler, der aus dem Netzmodell die Verhaltensbeschreibungsvorlage BDF generiert und dem MPPN-Simulator zur Verfügung stellt, wurde entwickelt. Die Erweiterungen, die im Rahmen dieser Arbeit an der graphischen Benutzeroberfläche gemacht wurden, werden im nächsten Abschnitt genauer erläutert.

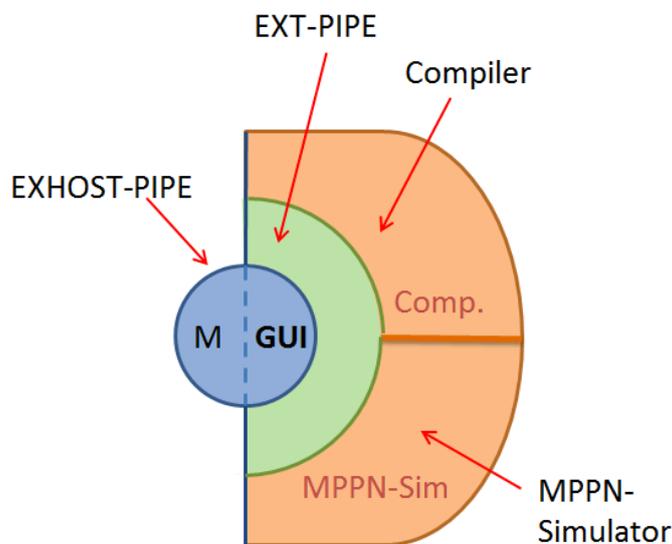


Abbildung 5.2.2: Struktur der Entwicklungsumgebung

Teile der Erweiterung betreffen die graphische Bedienoberfläche (Abbildung 5.2.3). Es wurden Menüpunkte für die Erstellung und Aktivierung von Partikelparametern und Konfigurationen hinzugefügt. Auch die allgemeine Bedienung wurde verbessert, indem Eingabefelder mit sinnvollen oder bereits eingegebenen Werten vorinitialisiert wurden und Anwenderfehler verhindert oder zumindest abgefangen wurden. Die nebenläufigen Simulationsalgorithmen wurden synchronisiert und können wahlweise automatisch oder schrittweise ausgeführt werden. Die Animation wurde um den XARA-Modus erweitert, bei der eine Textdatei mit Beobachtungen als Grundlage für die Simulation eines Partikel-Petri-Netzes dient. In Bezug auf Bedingungen an den Transitionen, wurden weitere logische Operationen definiert. Zur Entwicklung von hierarchischen Partikel-Petri-Netzen wurden sogenannte Fusion-Sets implementiert. Diese Erweiterungen erlauben, dass Systemverhalten in Bezug auf Sicherheitsgrenzen modelliert bzw. betrachtet werden können. Syntax- und Konsistenzüberprüfungen werden vom Editor während des gesamten Erstellungsprozesses

eines Netzmodells durchgeführt. Dabei stellt der Editor u. a. sicher, dass Stellen und Transitionen nicht mit unerlaubten Kanten verbunden werden.

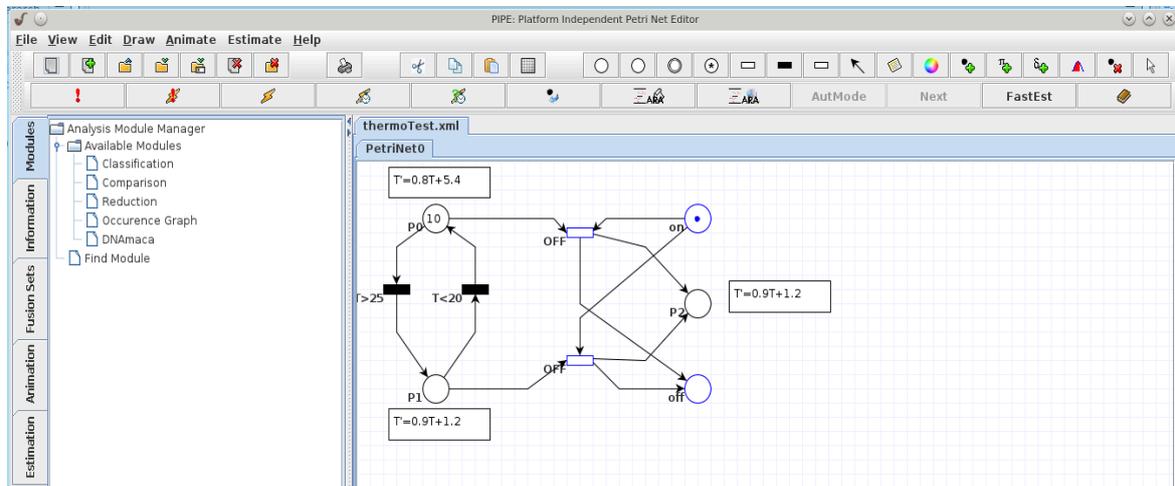


Abbildung 5.2.3: Graphische Bedienoberfläche von EXT-PIPE

Vor dem Start einer Simulation müssen die Art der Simulation (on-line oder off-line) und weitere Simulationsparameter, wie etwa der Startzustand und die maximale Simulationszeit, festgelegt werden. Im Folgenden erfolgt eine nähere Erläuterung der wichtigsten Erweiterungen:

a) Zusätzliche Bedingungen für numerische Transitionen

Die existierenden Transitionen im EXHOST-PIPE waren auf die mathematischen Operanden „<“, „>“ und „=“ und damit nur auf einfache Bedingungen zur Überprüfung beschränkt. Im Rahmen dieser Arbeit wurden neue Arten von Transitionen eingeführt, welche den Vergleich der Partikelwerte innerhalb von Intervallen als auch eine vektorielle Beschreibung einer Abfolge von Bedingungen ermöglichen. Hierfür wurden vier weitere Bedingungen hinzugefügt:

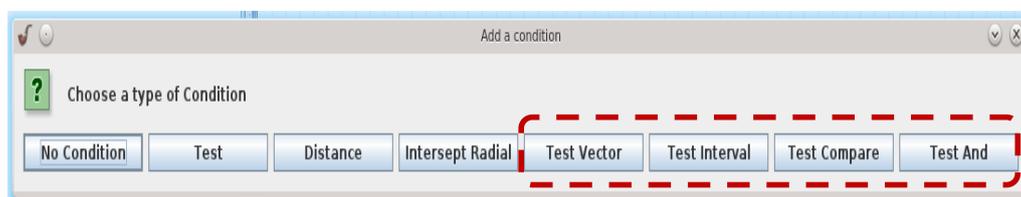


Abbildung 5.2.4: Erweiterungen an numerischen Transitionen

- **Test And** feuert eine numerische Transition nur, wenn der Partikel alle eingetragenen Bedingungen erfüllt. Mit „Test And“ ist es möglich, beliebig viele Bedingungen für eine Transition mit „Und-Verknüpfung“ zu erzeugen.

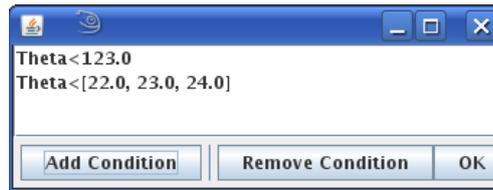


Abbildung 5.2.5: Test-And Bedingung

- **Test Compare** ist nur für Partikel-Petri-Netze mit mehreren Partikelparametern einsetzbar. In einem System können Interdependenzen zwischen den einzelnen kontinuierlichen Parametern bestehen, so dass beispielsweise ein Parameter wertemäßig immer unterhalb eines anderen liegen muss. Hiermit lässt sich eine solche Relation zwischen zwei Parameterwerten eines Partikels abbilden.

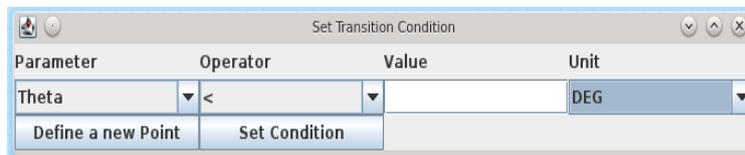


Abbildung 5.2.6: Test-Compare Bedingung

- **Test Interval** erlaubt einen Wertebereich festzulegen, in dem ein Partikelparameter sich befinden muss, um von dieser Transition gefeuert werden zu können. Dadurch ist es möglich, für einen Parameter verschiedene Wertebereiche festzulegen, in die jeweils unterschiedliche Transitionen des Netzes feuern. So können Switch-Case ähnliche Strukturen innerhalb eines Netzes realisiert werden.

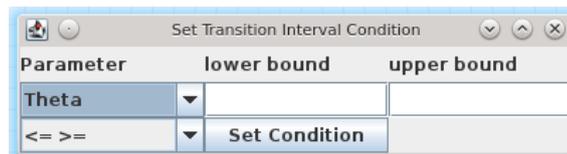


Abbildung 5.2.7: Test-Interval Bedingung

- **Test Vector**: Ein System kann über Parameter verfügen, die zu unterschiedlichen Zeitpunkten unterschiedliche Werte annehmen dürfen. Zur Modellierung dieses Umstands wurde diese Bedingung als Erweiterung implementiert. Sie prüft lediglich, ob ein Partikelparameter größer, kleiner oder gleich einem vorgegebenen Wert ist, aber genau dieser Wert lässt sich für jeden Simulationsschritt einzeln vorgeben.

```

>Transitions 5
[T=[26..28]][NUM][V] [Theta>[31.0 32.0 31.5 32.5 33.5 33.5 34.0 34.5 35.0 34.5
35.0 35.5 34.5 34.0 33.5 33.0 32.5 32.5 32.0 31.5 31.5 31.5 31.5]]
[T=[20..22]][NUM][V] [Theta<[29.0 30.0 29.5 30.5 31.5 31.5 32.0 32.5 33.0 32.5
33.0 33.5 32.5 32.0 31.5 31.0 30.5 30.5 30.0 29.5 29.5 29.5 29.5]]
[OFF][SYM][OP_Mode = 0.0]
[OFF][SYM][OP_Mode = 0.0]
[ON][SYM][OP_Mode = 1.0]

```

Abbildung 5.2.8: Test-Vector Bedingung

b) *Fusion Sets*

Damit komplexe Netze modular aufgebaut werden können, wurden so genannte Fusion-Sets implementiert (Abbildung 5.2.9). Diese erlauben eine Aufteilung in Unternetze (Abbildung 5.2.10), so dass die Partikel von einem Modul in ein anderes Modul propagieren können und damit eine Art Nachrichtenaustausch zwischen den verschiedenen Teilen des Gesamtsystems ermöglichen.

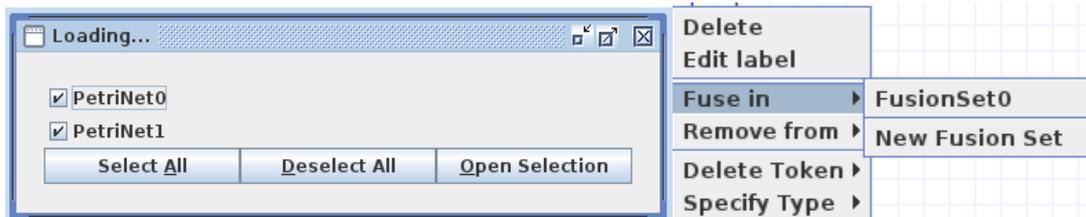


Abbildung 5.2.9: Fusion-Sets für hierarchische Petri-Netze

Dabei gilt, dass jedes Fusion-Set nur maximal eine numerische Stelle pro Subnetz enthalten darf. Gelangt ein Partikel in diese Stelle des Fusion-Sets, so wird er sofort in alle anderen Stellen des Fusion-Sets dupliziert.

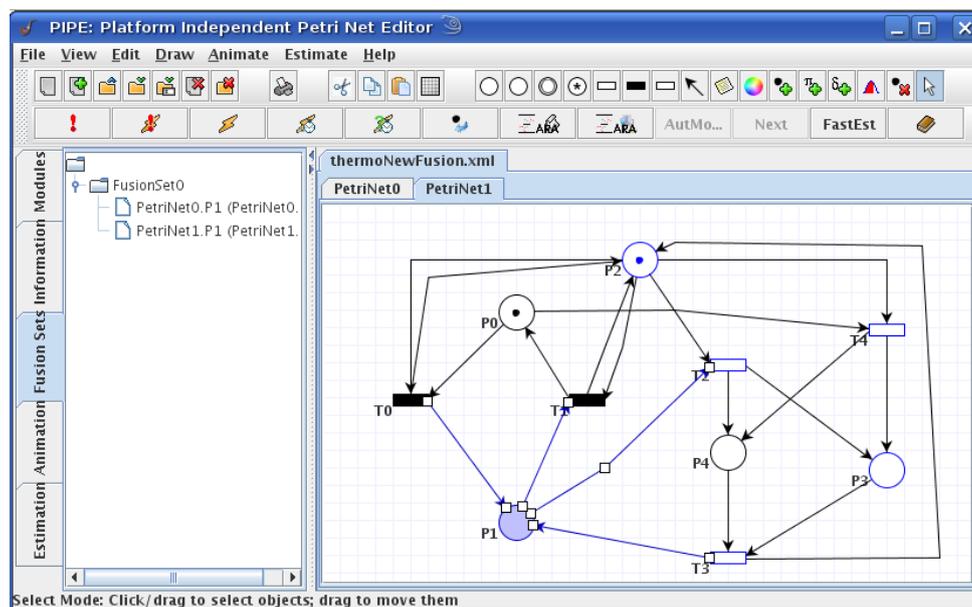


Abbildung 5.2.10: Austausch von Token zwischen hierarchischen Petri-Netzen

Das Hauptaugenmerk liegt im Aufbau der Petri-Netzmodelle, nach einer verschachtelten Struktur, wie die RNBC-Struktur bzw. einer verschachtelten Zerlegung, wie in Kobra. Abbildung 5.2.11 zeigt die Möglichkeiten der Implementierung von Fusion-Sets. Subnet₂ ist ein Unternetz von Subnet₁. Gelangt ein Token in eine Stelle eines Fusion-Sets, so erhalten beide Stellen an beiden Netzen diesen Token im Fall einer „Duplikation“. Im Fall eines Sprungs wird der Token dem Unternetz überführt und erscheint nicht mehr im Obernetz. Bei

der Duplikation wird es zwischen getrennter und verbundener Duplikation unterschieden. In der getrennten Duplikation, wenn ein Token eines Fusion-Sets entfernt wird, bleibt er in den anderen Stellen des Fusion-Sets erhalten. In der verbundenen Duplikation verlieren alle Stellen des Fusion-Sets ihren Token.

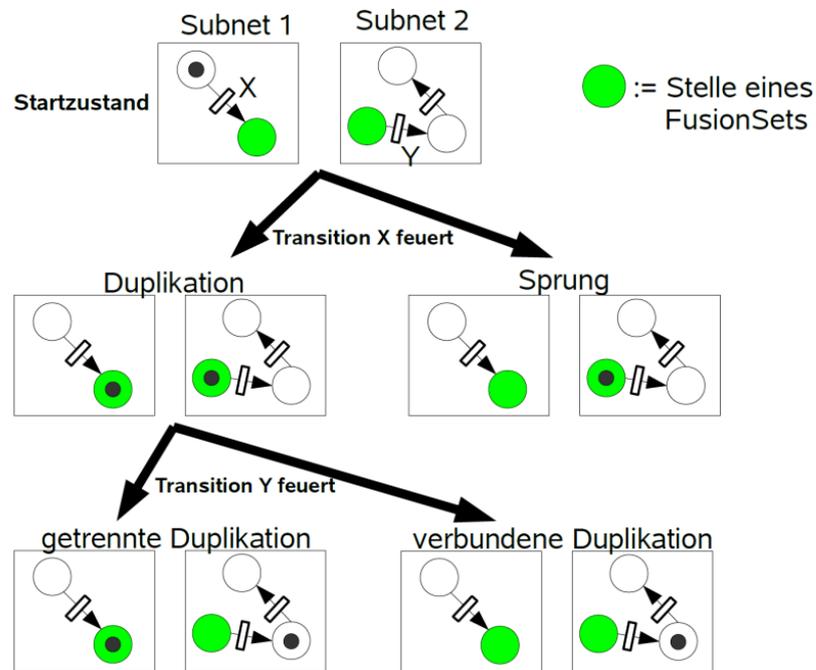


Abbildung 5.2.11: Möglichkeiten der Implementierung von Fusion-Sets

c) Realisation eines zeitvarianten Eingangs

Da Systeme in der Regel über einen sich ändernden Eingang verfügen, mussten die Differenzgleichungen der Stellen erweitert werden. Hierdurch können verschiedene, mit Leerzeichen getrennte Werte für B eingegeben werden, die ähnlich dem oben beschriebenen „Test Vector“ bearbeitet werden. Um dem Programm die Unterscheidung zwischen einem einzelnen Wert und mehreren zu vereinfachen, muss bei letzterem ein “<” vorangestellt werden.



Abbildung 5.2.12: Differenzgleichung mit zeitvariantem Eingang

d) Off-line Simulation mit einer Observationsdatei

Im Simulationsmodus kann über „Launch Estimation from File“ eine Datei mit off-line aufgezeichneten Daten ausgewählt werden und für die Simulation der Modelle verwendet

werden. Eine Simulation kann hier manuell oder automatisch ablaufen, mit einer Anzeige aller (Zwischen-) Ergebnisse im linken Teilfenster des Programms.

e) *On-line Simulation über Sockets*

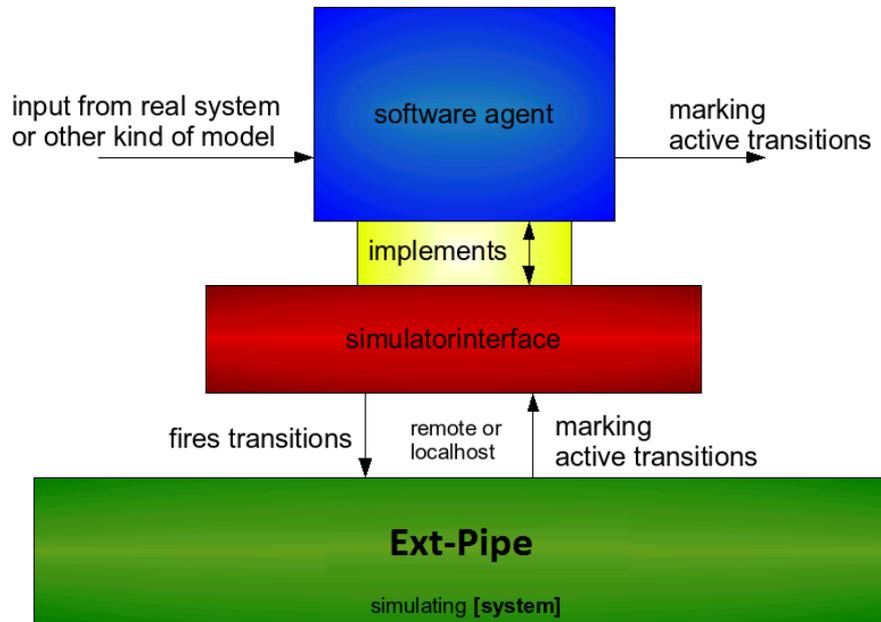


Abbildung 5.2.13: Implementierung der Schnittstellen für socketsbasierte Simulation

Für die Kommunikation mit einer externen Hardware und um eine on-line Simulation durchzuführen wurde ein Software-Modul (Software Agent) programmiert (Abbildung 5.2.13). Dieses erlaubt eine Sockets-basierte Simulation und schickt Daten über den aktuellen Systemzustand über Simulator-Schnittstellen (Simulator Interfaces). Diese verbinden sich zu einem beliebigen Hostrechner im Netzwerk, unter Eingabe von IP und Port (Abbildung 5.2.14). Das Protokoll entspricht dabei dem Format der Observationsdatei.

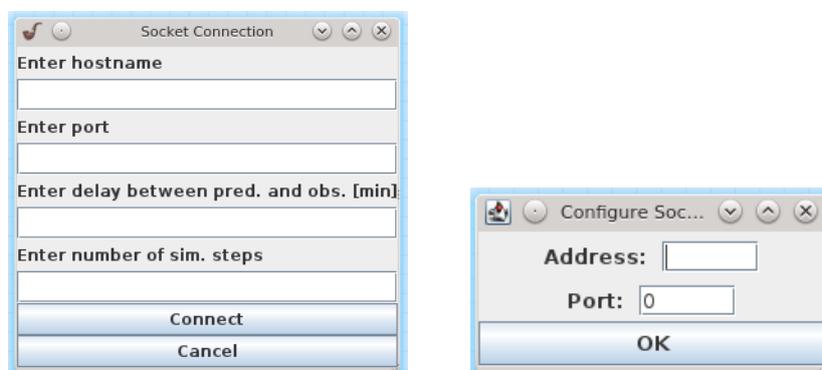


Abbildung 5.2.14: Socketverbindung und -konfiguration

Beobachtungen werden über Server durch Aktivierung des Modus „Launch Estimation from a Server“ geliefert. Bei Systemen mit langsamer Reaktion der Dynamik (z. B. eine

Temperaturänderung) kann zwischen Schätzung und Beobachtung eine Wartezeit eingebaut werden. Abbildung 5.2.15 zeigt die GUI des EXT-PIPEs während eines on-line Experiments mit dem realen Temperatursystem.

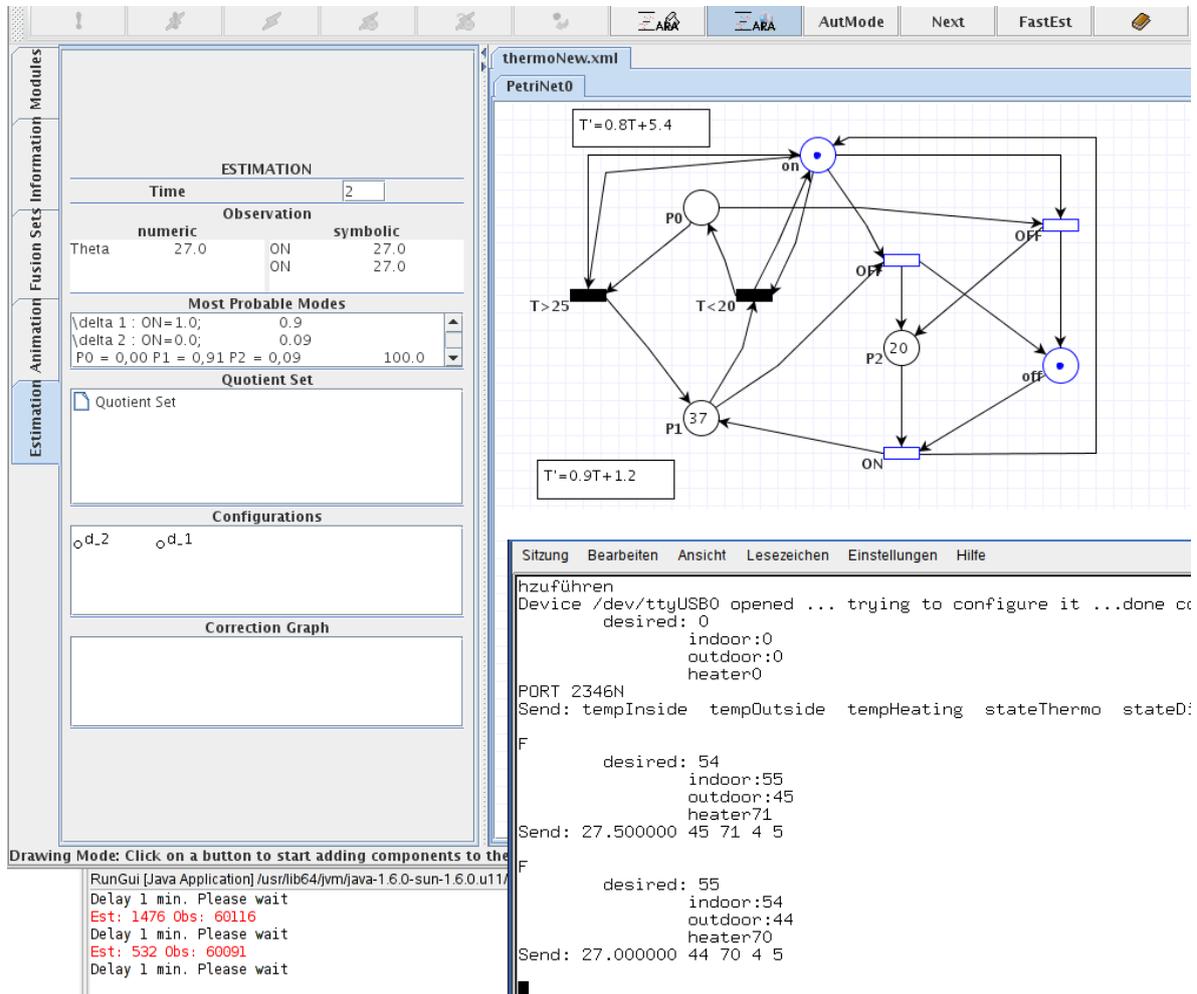


Abbildung 5.2.15: Echtzeitsimulation mit dem realen System

Es wird eine Referenztemperatur vom Nutzer definiert. Gemessen wird die Innen- und Außentemperatur. Der Regler passt die Leistung der Heizplatte an, um die gewünschte Temperatur zu erreichen. Gleichzeitig wird der hybride Systemzustand geschätzt.

f) *Erweiterungen bzw. Verbesserungen im Funktionenmenü*

Unter dem Menüpunkt „Estimate“ können die Parameter bzw. die Konfigurationen unter „Particle Parameters“ bzw. „Configuration Parameters“ ausgewählt und direkt im Netz benannt werden. Die Differenzgleichungen an den numerischen Stellen können durch Eingabe der A-Matrix und B-Vektor definiert werden. Numerische Bedingungen werden den numerischen Transitionen mit „Set Condition“ zugewiesen. Bei symbolischen Transitionen können mit „Set Effect“ mehrere Konfigurationen mit verschiedenen Effekten eingestellt

werden. Parametern der Unsicherheit, wie Rauschwerte, werden unter dem Menüpunkt „Estimate“ als Observation Noises bzw. Prediction Noises eingegeben. EXT-PIPE steht mit der Beendigung dieser Arbeit als eine quelloffene verbesserte Version von EXHOST-PIPE für die Simulation und Überwachung mittels Partikel-Petri-Netzen, sowohl on-line (mit realem System) als auch off-line (aus Observation-Datei), zur Verfügung. Abbildung 5.2.16 und Abbildung 5.2.17 zeigen die Simulationen des Temperaturregelungssystems (on- und off-line) mittels EXT-PIPE. Die Ergebnisse zeigen die Referenzwerte („+“ in rosa), bezogen auf die rechte Achse (Temperaturwerte in °C) und die Wahrscheinlichkeiten der diskreten Zustände (%-Werte) auf der linken Achse. Bei steigender Temperatur wird der Zustand „Heating“ als höchstwahrscheinlich geschätzt. Sinkt die Temperatur, sind die Zustände „Cooling“ und „OFF“ am wahrscheinlichsten, wobei in diesen Fällen der Zustand „Cooling“ selektiert wird, da das System nicht von außen abgeschaltet wurde.

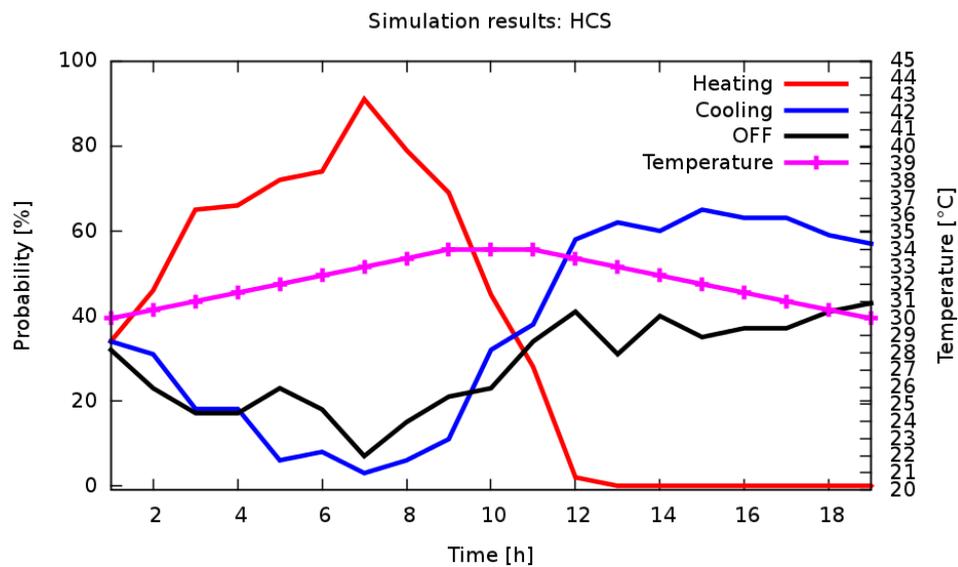


Abbildung 5.2.16: Ergebnisse bei einer off-line Simulation

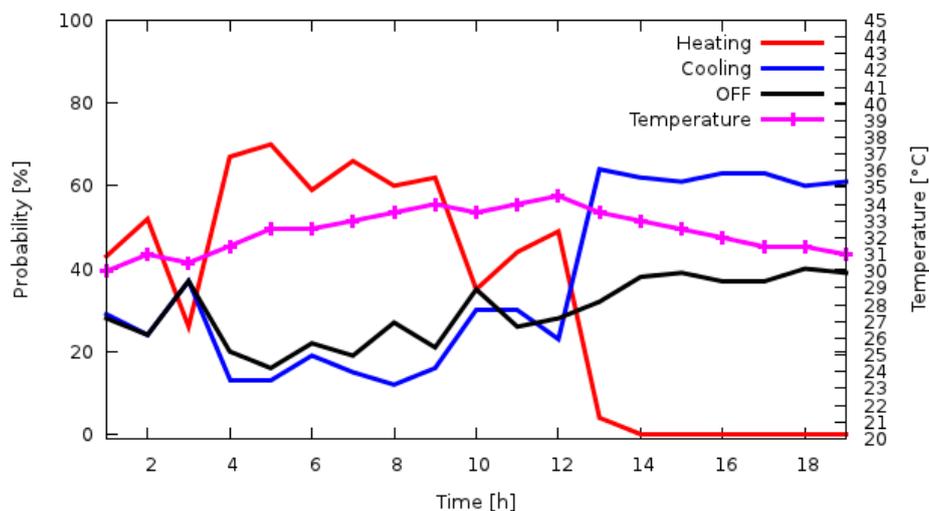


Abbildung 5.2.17: Ergebnisse bei einer on-line Simulation

5.3 Der Netzcompiler und Export in XML-Format

Die Aufgabe des Netzcompilers besteht darin, die graphische Darstellung eines MPPN-Modells aus dem Editor in einem Dateiformat für Petri-Netz-Werkzeuge zu beschreiben. Diese Beschreibung soll einerseits erweiterbar sein, dass noch nicht repräsentierte Petri-Netz-Typen leicht integriert werden können, und andererseits soll diese Datei ohne Konvertierungen für den Menschen verständlich sein. Diese Anforderung erfüllt die Dokumentenbeschreibungssprache XML. Die Verwendung von Standard Beschreibungssprachen erleichtert allgemein den Austausch von Petri-Netzen zwischen verschiedenen Werkzeugen, da die Struktur einer Petri-Netz-Datei und die Bedeutung einzelner Teile (Stellen, Transitionen und Kanten) festgelegt sind. In dieser Arbeit wird die textuelle Repräsentation von Petri-Netzen mit Informationen aus der Spezifikation auf einheitliche Weise ergänzt und als BDF für die on-line Überwachung des Systems eingesetzt. Abbildung 5.3.1 zeigt die Hauptbestandteile der entwickelten Umgebung, von der Erstellung des Systemmodells über die Generierung des BDF bis zur Ergebnisauswertung und -aufbereitung.

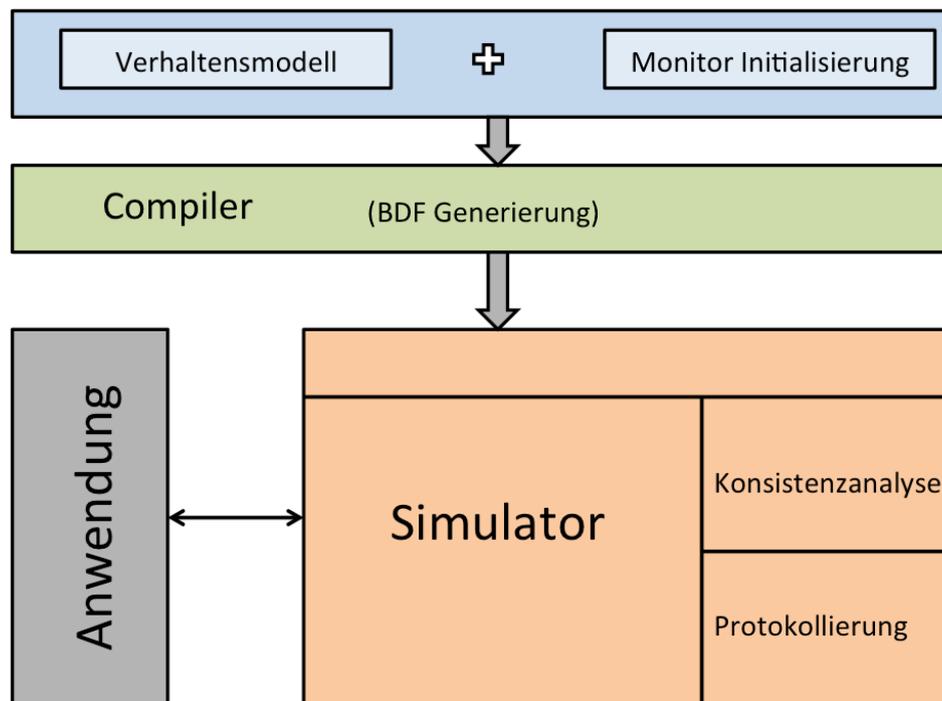


Abbildung 5.3.1: Schritte der Überwachungsentwicklung und -simulation

Der Netzcompiler ist in C++ implementiert. Abbildung 5.3.2 zeigt das Sequenzdiagramm des Netzcompilers mit den wichtigsten Funktionen und allen Objektaufufen, um ein Netzmodell „Net“ zu einer Textdatei „filename.txt“ zu konvertieren und diese dem Simulator „SimNet“ zur Verfügung zu stellen.

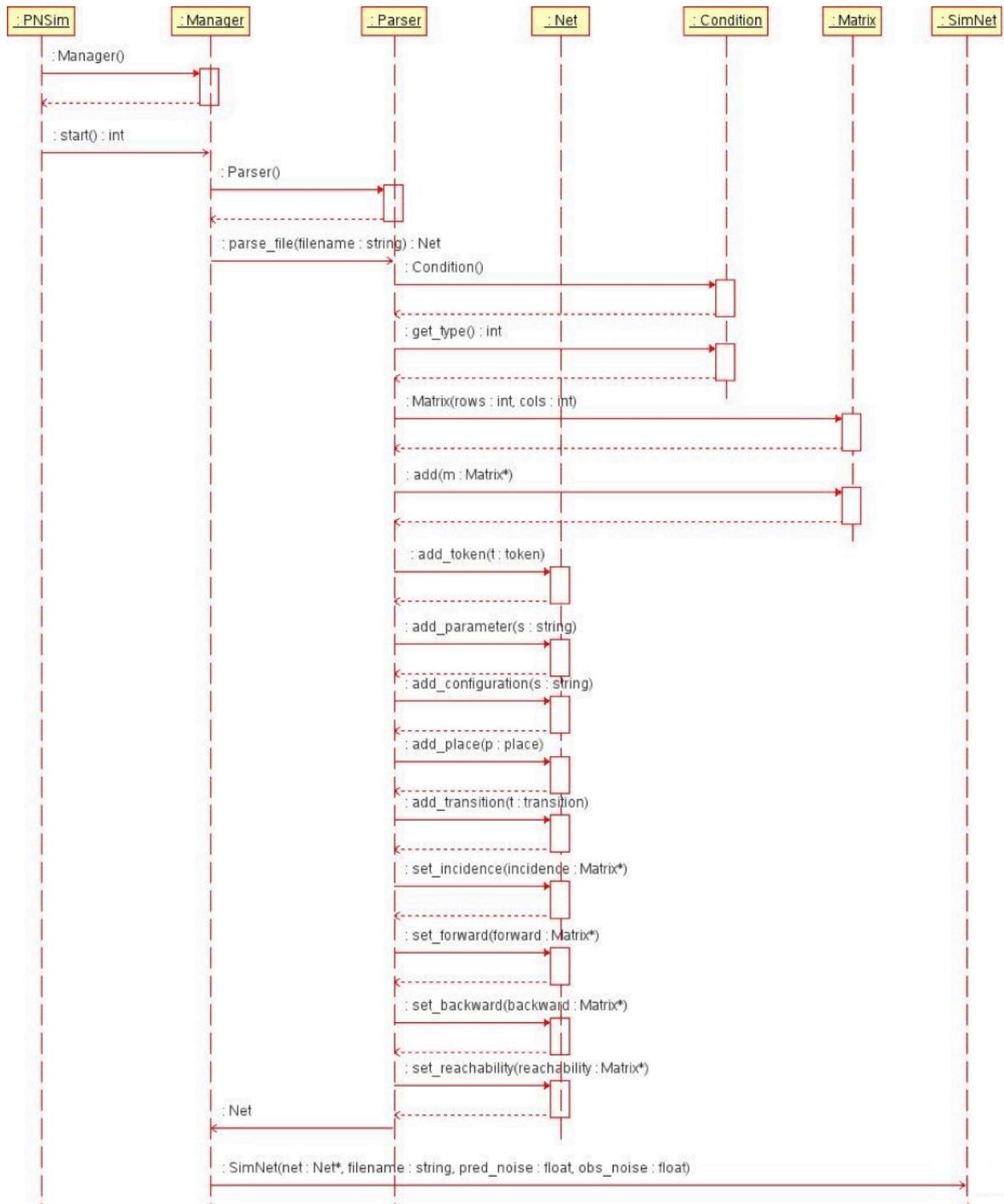


Abbildung 5.3.2: Sequenzdiagramm zur Erstellung einer Netzdatei

5.4 Der MPPN Simulator

Der MPPN-Simulator „PNSim“ wurde unter Linux in der Programmiersprache C++ entwickelt. Er bekommt als Eingang die XML-basierte textuelle Beschreibung des zu simulierenden Überwachungsmodells und ermöglicht sowohl off-line als auch on-line Simulationen. In einer off-line Simulation werden die gemessenen Daten, wie beim EXT-PIPE, aus einer Datei „Observation-File“ ausgelesen und zur Verfügung gestellt. Für eine on-

line Simulation muss eine geeignete Schnittstelle zum realen System geschaffen werden, und anhand eines Servers werden die realen Daten über einen Socket an den Simulator gesendet. Es werden die Systemzustände und die Residuen der kontinuierlichen Parameter zu jedem Zeitpunkt berechnet und zur Detektion von Inkonsistenzen im Prozess verwendet. Der MPPN-Simulator stellt eine Implementierung der MPPN-Regeln, die in Kapitel 3 erläutert wurden dar. Die einzelnen Prozesse der MPPN hybride Schätzung werden hier mittels Ausschnitte aus dem Pseudo-Code erklärt.

5.4.1 Variablen, Funktionen und Prozesse

a) Globale Variablen:

Variablen	Bedeutung
<pre>Token: struct { numeric: Boolean place_nr: Integer value: Float weight: Float } Place: struct { numeric: Boolean place_id: Integer prob : Float A: Matrix B: Matrix state: Integer } Transition: struct { numeric: Boolean transition_id: Integer condition: Condition } token_class : struct { place_nr: Integer particles: Vector <Token> weight: Float }</pre>	<pre>//Structure Token for Token and Particles //true if numeric, false if symbolic //Place number of token or particle //Value of particle, don't care for token //Weight of Particle, don't care for token //Structure for Places //true if numeric, false if symbolic //Place number //Probability of Place //A-matrix of differential equation (only numeric places) //B-matrix of differential equation (only numeric places) //State of configuration (only symbolic) //Structure for Transitions //true if numeric, false if symbolic //Transition number //Condition of Transition (only numeric transitions) //Structure Token Class //Place number //Particles in class //Weight of the class</pre>

<pre> token_pair: struct { class: token_class config: Configuration } Simsteps: Integer obs_noise: Float pred_noise: Float P:Vector<Places> T:Vector<Transition> M_{k k}: Vector <Token> M_{k k}^{Terminal}: Vector <Token> M̂_{k+1 k}: Vector <Token> M̂_{k+1 k+1}: Vector <Token> O^S: Vector <Float> O^N: Vector <Float> R_G: Matrix count_num: Integer Configurations: String </pre>	<pre> //Structure Token pair //Token class //Configuration value //Number of simulation steps to be done //Observation Noise //Prediction noise //Vector of struct Place //Vector of struct Transition //Vector of struct Token //Terminal Marking //Estimated marking at time k+1 knowing observation at time k //Estimated marking at time k+1 //Symbolic observation data //Numeric observation data //Rechability matrix //Number of initial particles //All configurations of the net } </pre>
---	---

b) Globale Funktionen:

Funktionen	Bedeutung
V.size()	//Returns size of Vector V
V(i:integer)	//Returns entry number i of VectorV
V.clear()	//Delete all entrys of Vector V
test_condition(t_nr:integer,t:Token)	//Returns true if condition is satisfied, else false
move_particle(p_nr:integer,t_nr:integer)	//Move particle from preplace to postplace
copy_particles(p1_nr:integer,p2_nr:integer)	//Copy particles from preplace to postplace
copy_token(p1_nr:integer, p2_nr:integer)	//Copy token from preplace to postplace
norm(particles : V ector <T oken>)	//Norm weights of particles to 1
sort(particles : V ector <T oken>)	//Sort particles by weight
class(particles : V ector <T oken>)	//Classify particles
getP articles(particles : V ector <T oken>)	//Returns all particles, without token
make_pairs (t_classesVector <token_class>)	//Returns pairs of Configurations and Token Classes
max_prob_s_place()	//Returns symbolic place with highest probability
max_prob_n_place()	//Returns numerical place with highest probability

c) MPPN Prozesse:

Die Zustandsschätzung mittels MPPN besteht aus einer *Prädiktion (Prediction)* und einer *Korrektur (Correction)*. Der Prädiktionsschritt wiederum ergibt sich aus der *Prädiktion der Petri-Netz-Markierung (Marking Prediction)* und der *Prädiktion der Partikelwerte (Particle Prediction)*. Das Ergebnis dieses Schritts ist eine Vorhersage aller nächsten möglichen Zustände des Systems. Im Korrekturschritt (*Correction*) wird diese Vorhersage, entsprechend einer vom System vorgenommenen neuen Beobachtung, aktualisiert. Algorithmus 1 zeigt die Implementierung dieser Prozesse, die in jedem Simulationsschritt aufgerufen werden:

Algorithm 1	Simulation Step
1:	while Simsteps > 0 do
2:	Marking Prediction;
3:	Particle Prediction;
4:	Correction;
5:	Simsteps ← Simsteps -1;
6:	end while

5.4.2 Prädiktion der Markierung (Marking Prediction)

In diesem Schritt wird aus der Markierung M_k zum Zeitpunkt k die Endmarkierung M_k^T (Terminal Marking) ermittelt. M_k^T enthält alle möglichen Folgemarkierungen für den Zeitpunkt $k+1$. Am Anfang des Algorithmus 2 werden alle Transitionen T auf Pseudo-Feuerbarkeit (*pseudo_fire()*) überprüft.

Algorithm 2	Marking Prediction
Input : $M_{k k}$	
Output : $M_{k k}^{\text{Terminal}}$	
en ← 0 : Integer	
T_{fired} , T_{enabled} : Vector<Integer //Transition IDs	
last_fire ← T.size()+1 : Integer //Init with not assigned	
Trans.ID ensuring	
fired ← false : Boolean	
for i = 0 to T.size() -1 do	
if pseudo_fire(T(i)) = true then	
$T_{\text{enabled}} \leftarrow T(i)$	
end if	
end for	
while $T_{\text{enabled}} > 0$ do	
if $T_{\text{enabled}}. \text{size}(), > \text{en}+1$ then	
if last_fire = $T_{\text{enabled}}(\text{en})$ then	
en ← en+1	
else	
en ← 0	
end if	
end while	

```

else
    en ← 0
end if

    fire (T_enabled (en))
    last_fire ← T_enabled(en)
    T_fired ← T_enabled (en)
    T_enabled.clear
For i = 0 to T.size() -1
If pseudo_fire(T(i)) == true then
    If T(i) in T_fired then
        fired ← true
    end if
    if fired = false then
        T_enabled ← T(i)
    else
        fired ← false then
    end if
end if
end for
end while

```

Dieser Schritt des *pseudo_fire()* wird anhand des Algorithmus 3 durchgeführt: Sei ${}^{\circ}T_j$ die Menge der Eingangsstellen (*Preplace*) für die Transition T_j , und T_j° ist die Menge der Ausgangsstellen (*Postplace*). Eine numerische bzw. symbolische Transition T_j ist pseudo-feuerbar (*fire_enabled==True*), falls alle ihre numerischen bzw. symbolischen Eingangsstellen belegt sind, d. h.:

$$\forall P^i \in {}^{\circ}T_j : m_k(p_i) \geq Pre(p_i, T_j). \quad (5.4.1)$$

Eine hybride Transition T_j , ist nur feuerbar, wenn sowohl ihre numerische als auch symbolische Stellen im Vorbereich belegt sind:

$$\forall p_i \in {}^{\circ}T_j : m_k(p_i) \geq Pre(p_i, T_j), p_i \in P^S \wedge m_k(p_i) \geq 0, p_i \in P^N \quad (5.4.2)$$

Algorithm 3**Pseudo Fire**

```

Input : t : Transition, Mk|k, Preplace : Vector <Integer>
Output : fire_enabled : Boolean
Preplace : Vector <Integer> //Vector with Pre-Place IDs of
Transition t
count[Preplace.size()] : Array of Integer & //Save count of
particles/token of all pre-places

tmp_cnt ← 0 : Integer //Temporary variable
for i = 0 to Preplace.size() -1 do
    count[i] ← 0

```

```

    for j = 0 to Mk|k.size() - 1 do
        if Mk|k(j).place_nr == Preplace(i) then
            count[i] gets count[i] + 1
        end if
    end for
end for
for i = 0 to Preplace.size() - 1 do
    if count[i] # 0 then
        tmp_cnt ← tmp_cnt + 1
    end if
end for
if temp == Preplace.size() then return true

else
    return false
end if

```

Algorithmus 3: Überprüfung der Pseudo-Feuerbarkeit einer Transition

Nach der Ermittlung der pseudo-feuerbaren (*fire_enabled*) Transitionen, werden die Bedingungen für die Feuerbarkeit überprüft (Algorithmus 4). Seien p_n bzw. p_s die numerische bzw. symbolische Eingangsstellen einer Transition und seien p'_n und p'_s die Ausgangsstellen. Das Feuerbarkeitskonzept der MPPN sieht wie folgt aus:

Regel 5.4.1: Eine feuerbare numerische Transition T_j feuert, wenn Partikel $\pi \in \Pi_F$ existieren, die die Bedingungen der numerischen Transition erfüllen $\Omega_N(t_n)(\pi) = 1$. Das numerische Feuern verwendet das Konzept des:

- klassischen Feuerns mit den Token in den numerischen Stellen (den Partikeln), d. h. nur diejenigen Partikel, die die numerische Bedingung erfüllen, wandern (*move_particle()*) von einer Stelle p_n zu p'_n :

$$\forall p_n \in {}^\circ T_j \cap P^N, \forall p'_n \in T_j \circ \cap P^N : \Pi(p_n) = \Pi(p_n) \setminus \Pi_F \quad ; \quad \Pi(p'_n) = \Pi(p'_n) \cup \Pi_F \quad (5.4.3)$$

- Pseudo-Feuerns mit den Token in den symbolischen Stellen (Konfigurationen), d. h. die Konfigurationen der Stellen im Vorbereich werden in die Stellen im Nachbereich kopiert (*copy_token()*), ohne sie aus den Stellen im Vorbereich zu löschen:

$$\forall p_s \in {}^\circ T_j \cap P^S, \forall p'_s \in T_j \circ \cap P^S : \Delta(p_s) = \Delta(p_s) \quad ; \quad \Delta(p'_s) = \Delta(p'_s) \cup \Delta(p_s) \quad (5.4.4)$$

Regel 5.4.2: Wenn eine feuerbare symbolische Transition T_j feuert, werden sowohl die Partikel als auch die Token von den Stellen im Vorbereich in die Stellen im Nachbereich kopiert (*copy_token()*, *copy_particles()*), nach den folgenden Regeln:

$$\Pi(p_n) = \Pi(p_n); \quad \Pi(p'_n) = \Pi(p'_n) \cup \Pi(p_n) \quad (5.4.5)$$

$$\Delta(p_s) = \Delta(p_s); \quad \Delta(p'_s) = \Delta(p'_s) \cup \Delta(p_s) \quad (5.4.6)$$

Dadurch ergibt sich aus der alten Markierung $M(p) = \{\Pi(p_n), \Delta(p_s)\}$ eine neue Markierung $M(p') = \{\Pi(p'_n), \Delta(p'_s)\}$ (siehe Algorithmus 2: **Input** $M_{k|k}$, **Output** $M_{kk}^{Terminal}$).

Algorithm 4	Fire
<p>Input: t: Transition, Preplace: Vector<Integer>, Postplace: Vector<Integer>, $M_{k k}$</p>	
<pre> If t.numeric == 1 then for i = 0 to Preplace.size() -1 do If Preplace(i).numeric == true then for j = 0 to M_{k k}.size() -1 do if M_{k k}(j).place_nr == Preplace(i) then if test_num_condition(t, M_{k k}(j)) == true then move_particle(Preplace(i),t) end if end if end for else for j = 0 to M_{k k}.size() -1 do if M_{k k}(j).place_nr == Preplace(i) then for k = 0 to Postplace.size() -1 do if Postplace(k).numeric == false then copy_token(Preplace(i),Postplace(k)) end if end for end if end for end if end for else for i = 0 to Preplace.size() -1 do if Preplace(i).numeric == true then for j = 0 to M_{k k}.size() -1 do if M_{k k}(j).place_nr == Preplace(i) then for k = 0 to Postplace.size() -1 do if Postplace(k).numeric == true thrn copy_particles(Preplace(i),Postplace(k)) end if end for end if end for else for j = 0 to M_{k k}.size() -1 do if M_{k k}(j).place_nr == Preplace(i) then </pre>	

```

                                for k = 0 to Postplace.size() -1 do
                                    if Postplace(k).numeric == false then
                                        copy_token(Preplace(i),Postplace(k))
                                        end if
                                    end for
                                end if
                            end for
for i = 0 to Preplace.size() -1 do
    If Preplace(i).numeric == true then
        for j = 0 to Mk|k.size() -1 do
            if Mk|k (j).place_nr == Preplace(i) then
for k = 0 to Postplace.size() -1 do
                    if Postplace(k).numeric == false then
                        if test_csymb_ondition(t, Mk|k (j)) == true then
                            move_particle(Preplace(i),t)
                            end if
                        end if
                    end if
                end for
            else
                for j = 0 to Mk|k.size() -1 do
                    if Mk|k (j).place\_nr == Preplace(i) then
                        for k = 0 to Postplace.size() -1 do
                            if Postplace(k).numeric == false then
if test_symb_condition(t, Mk|k (j)) == true then
                                    copy_token(Preplace(i),Postplace(k))
                                end if
                            end if
                        end for
                    end if
                end for
            end if
        end for
    end if
end for
end if

```

Regel 5.4.3: Eine hybride Transition T_j feuert den symbolischen Teil, völlig unabhängig vom numerischen Teil, solange die symbolischen Token $\delta \in \Delta_F$ die symbolische Bedingung an die hybride Transition erfüllen $\Omega_S(t_h)(\delta) = 1$. Allerdings bildet dieser Teil des Feuerns eine notwendige Bedingung für das Feuern des numerischen Teils. Dies ist für die Berücksichtigung der Wechselwirkung beider Teile der Dynamik wichtig. Ist diese Voraussetzung erfüllt und erfüllen einige Partikel $\pi \in \Pi_F$ die numerische Bedingung $\Omega_N(t_h)(\pi) = 1$, dann werden gleichzeitig die Partikel verschoben und die Token kopiert. Es gilt:

$$\forall p_n \in {}^\circ T_j \cap P^N, \forall p'_n \in T_j^\circ \cap P^N \text{ und } \Pi_F \subseteq \Pi(p_n)$$

$$\Pi(p_n) = \Pi(p_n) \setminus \Pi_F \quad ; \quad \Pi(p'_n) = \Pi(p'_n) \cup \Pi_F \quad (5.4.7)$$

$$\Delta(p_s) = \Delta(p_s) \quad ; \quad \Delta(p'_s) = \Delta(p'_s) \cup \Delta(p_s) \quad (5.4.8)$$

5.4.3 Prädiktion der Partikel (Particle Prediction)

Die Entwicklung der Partikel nach einem kontinuierlichen Modell (Algorithmus 5) hängt von deren Lage im Petri-Netz ab, d. h. vom numerischen Teil der Makromarkierung ($M_{kk}^{\text{Terminal}}(j).numeric$). Das Ergebnis dieses Schrittes ist eine Vorhersage (prediction), sowohl der neuen Werte der Partikel als auch die Folgemarkierung des Petri-Netzes.

Algorithm 5	Particle Prediction
<p>Input : $M_{k k}^{\text{Terminal}}$ Output : $M_{k+1 k}$</p> <pre> For i = 0 to P.size() -1 do If P(i).numeric == true then For j = 0 to $M_{k k}^{\text{Terminal}}$.size() -1 do If $M_{k k}^{\text{Terminal}}(j).place_nr == P(i).ID$ AND ($M_{k k}^{\text{Terminal}}(j).numeric$) then $M_{k k}^{\text{Terminal}}(j).value \leftarrow P.A * M_{k k}^{\text{Terminal}}(j).value + P.B + pred_noise$ end if end for end if end for </pre>	

5.4.4 Korrekturschritt (Correction step)

Nach der Prädiktion erfolgt eine Korrekturphase, sowohl der geschätzten Markierung als auch der Partikelwerte. Algorithmus 6 zeigt die Implementierung dieses Korrekturschrittes, der aus einer numerischen und einer symbolischen Korrektur besteht:

a) Die numerische Korrektur betrifft nur die Partikel der Endmarkierung und ist in drei Schritte unterteilt: Partikel Gewichtung (Zeile 13), Resampling (Zeile 18) und Klassenbildung (Zeilen 19 und 20).

Algorithm 6	Correction
<p>Input: $\hat{M}_{k+1 k}$, O_k, R_G Output: $\hat{M}_{k+1 k+1}$</p> <pre> particles : Vector<Token> //All particles without tokens token : Vector<Token> //All token without particles t_classes : Vector<token_class> //Vector of token classes pairs : Vector<token_pair> //Vector of token pairs pair : token_pair //Token pair </pre>	

```

reachable : Boolean, //true if state consistent, else false
Mk|kS: Token, //symbolic token at time k
Mk+1|k+1S : Token, & //symbolic token at time k+1

is_last_state : Boolean
obs_ok : Boolean
  for i = 0 to M.size() - 1 do
    if M(i).numeric == true then

$$M(i).weight = \frac{1}{\sqrt{2\pi}} e^{-\frac{(M(i).value - O_K^N)^2 * obs\_noise}{2}}$$

    end if
  end for
  particles ← getParticles(M(i));
  norm(particles)
  particles ← resample(particles)
  particles ← sort(particles)
  t_classes ← class(particles)
  for i = 0 to P.size() -1do
    if P(i).numeric == true then
      P(i).prob ← 0
      For j = 0 to t_classes.size() -1do
        if t_classes(j).place_nr == P(i).place_id then
          P(i).prob = P(i).prob + t_classes(j).weight
        end if
      end for
    end if
  end for
  for i = 0 to P.size() -1 do
    if P(i).numeric == false then
      for j = 0 to t_classes.size() -1do
        pair ← (t_classes(j),P(i))
        if pair ∈ R_G then
          P(i).prob = P(i).prob + P(j).prob
        end if
      end for
    end if
  end for
  is_last_state=false;
  obs_ok=false;
  if OKS ∈ Configurations}
    if OKS == max_prob_s_place().State then
      for i = 0 to token.size() -1do
        if token(i).place_nr== max_prob_s_place().place_id
then

$$M_{k+1|k+1}^S \leftarrow \text{token}(i)$$

          obs_ok=true;
          is_last_state=false;
        end if
      end for
    else
      is_last_state=true;

```

```

        obs_ok=false;
    end if
else
    is_last_state=true;
    obs_ok=false;
end if
pairs ← make_pairs(t_classes)
if (is_last_state) then
    pair ← (max_prob_n_place, mk|kS)
    if pair ∈ RG then

        m̂k+1|k+1S ← [mk|kS, particles]
    else
        if (lobs_ok) then
        for i = 0 to token.size() -1do
            if token(i).place_nr== max_prob_s_place().place_id
                mk+1|k+1S ←token(i)
            end if
        end for
        end if
        for i = 0 to pairs.size() -1 do
            if pairs ∈ RG then
                reachable ← true
            else
                reachable ← false
            end if
        end for
        M̂k+1|k+1S ←[mk+1|k+1S, particles]
    end if
end if
end if

```

Algorithmus 6 ruft einen wichtigen Prozess auf (Zeile 18), nämlich den Resampling-Prozess. Das Ziel dieses Schrittes ist es, die Partikel mit geringem Gewicht zu löschen, da nur die Partikel mit der hohen Wahrscheinlichkeit den Systemzustand repräsentieren. Für das Resampling Algorithmus gibt es verschiedene Ansätze ([Dou05]). Algorithmus 7 zeigt den Pseudo-Code des hier vorgeschlagenen Resamplingalgorithmus, wobei $\hat{\Pi}_{k+1|k} \subseteq \hat{M}_{k+1|k}$ die sortierten Partikel und $\hat{\Pi}_{k+1|k+1}$ die Partikel nach dem Resampling sind.

Algorithm 7 Resampling

```

Input : particles : Vector<Token>, count_num
Output : new_particles : Vector<Token>
done ←false :Boolean // Random number between 0..1
c[particles.size()] : Array of Float // Accumulated weights
z : Float // Random number between 0..1
    c[0] ←particles(0).weight
    for l = 1 to particles.size() -1 do
        c[l] ← c[l-1] + particles(l).weight
    end for
    for j = 0 to count_num -1do

```

```

done ← false
while done == false do
  z ← random(){Random number between 0..1}
  for i = 0 to particles.size() -1 do
    if (c[i] < z) AND (c[i+1] ≥ z) then
      new_particles ← particles(i)
      new_particles.weight ←  $\frac{1}{\text{count\_num}}$ 
      done ← true
    end if
  end for
end while
end for
return new_particles

```

b) Die symbolische Korrektur vergleicht die Menge der Konfigurationen $\Omega(p_m)$, die sich in der höchstwahrscheinlichen symbolischen Stelle p_m befinden, mit den symbolischen Beobachtungen O_k^S und überprüft die Erreichbarkeit des hybriden Zustands $(p(\kappa_1), p_m)$. $p(\kappa_1)$ repräsentiert die Klasse mit der höchsten Gewichtung. Das Ergebnis dieses Entscheidungsalgorithmus ist die geschätzte symbolische Markierung $\hat{\Delta}_{k+1|k+1}$.

Algorithm 8	Decision Making
<p>Input : $\hat{\Delta}_{k+1 k}$</p> <p>Output : $\hat{\Delta}_{k+1 k+1}$</p> <pre> if ($\hat{M}_{k+1 k}(p_m) \subseteq \hat{\Delta}_{k+1 k}$) AND ($\hat{M}_{k+1 k}(p_m) \neq \emptyset$) then if $O_k^S == \Omega(p_m)$ then $\hat{\Delta}_{k+1 k+1} \leftarrow \hat{M}_{k+1 k}(p_m)$ else if $(P(\kappa_1), p_m) \in R_G$ then $\hat{\Delta}_{k+1 k+1} \leftarrow \Delta_{k k}$ else $\hat{\Delta}_{k+1 k+1} \leftarrow \hat{M}_{k+1 k}(p_m)$ end if end if else if $(P(\kappa_1), p_m) \in R_G$ then $\hat{\Delta}_{k+1 k+1} \leftarrow \Delta_{k k}$ else $\hat{M}_{k+1 k}(p_m) \leftarrow \delta_i$ { Generate Configuration } $\hat{\Delta}_{k+1 k+1} \leftarrow \hat{M}_{k+1 k}(p_m)$ end if end if return $\hat{\Delta}_{k+1 k+1}$ </pre>	

6 Fallstudie: Überwachung der Navigation autonomer mobiler Roboter in einer Indoor-Umgebung

Die vorherigen Kapitel befassten sich mit der Entwicklung und Implementierung eines Verfahrens für die Online-Überwachung hybrider dynamischer Systeme allgemein. In diesem Kapitel geht es um den Einsatz des Verfahrens für Autonome Mobile Roboter (AMR) speziell. Als Demonstrationssystem zur Überprüfung der Umsetzbarkeit des Verfahrens wird ein intelligenter Rollstuhl verwendet. Das Verfahren soll ebenfalls auf andere autonome Systeme übertragbar sein. In Abschnitt 6.1 werden die für diese Arbeit relevanten Grundlagen aus der Robotik eingeführt. Auf die Entwicklung und Implementierung der Überwachungsmodelle für den autonomen Rollstuhl wird in den Abschnitten 6.2 und 6.3 eingegangen. Die Modelle und die Ergebnisse dafür ([Mek11a]- ([Mek11b]) werden im Abschnitt 6.4 erläutert.

6.1 Arbeitsrelevante Grundlagen der Navigation mobiler Roboter

Zum intelligenten und selbstständigen Verhalten eines autonomen mobilen Roboters gehört die Fähigkeit zum Navigieren. Der Roboter muss in der Lage sein, autonom, kollisionsfrei und möglichst effizient zu einem Zielort zu fahren. Für die Navigation können folgende wesentliche Teilaufgaben unterschieden werden:

- Umgebungsmodellierung: Die Grundlage der Umgebungsmodellierung bildet eine Karte, die den gesamten Navigationsbereich des Roboters umfasst und eine Beschreibung aller darin bekannten statischen Hindernisse enthält.
- Selbstlokalisierung: Der Roboter muss seine eigene Position innerhalb der Umgebung bestimmen und während der Fahrt kontinuierlich nachverfolgen.
- Pfadplanung: Berechnung der besten Fahrtroute von der aktuellen Position zum Zielpunkt.

6.1.1 Umgebungsmodellierung

Eine Repräsentation der Umgebung durch Weltmodelle ist von zentraler Bedeutung für mobile Roboter und sehr wichtig für die Selbstlokalisierung und Wegplanung. Wenn der Roboter bestimmte Zielpunkte in der Umgebung anfahren soll, benötigt er dazu eine interne Repräsentation dieser Umgebung.

Man unterscheidet zwischen verschiedenen Methoden der Umgebungsrepräsentation. Ist die Umgebung des Roboters unbekannt, d. h. eine Karte ist nicht vorhanden, muss der Roboter

mit Hilfe von Sensoren (z. B. Ultraschall oder Laser) Informationen über seine Umgebung sammeln und eine Karte erstellen. Diese Situation wird „simultaneous localization and mapping (SLAM)“ genannt. Einige Lösungsansätze für dieses Problem wurden in [Dur06] beschrieben. Ist die Umgebung jedoch bekannt, kann man sie unter anderem mit einer geometrischen oder einer topologischen Karte darstellen. Die Wahl einer geeigneten Darstellung hängt im Wesentlichen von der Aufgabe und der dafür notwendigen Auflösung, Art und Anzahl der Hindernisse sowie der Größe des Roboters ab. Im Folgenden werden die verschiedenen Umgebungsmodelle vorgestellt:

a) Geometrisches Modell

Durch solche Karten lassen sich Umgebungen und Objekte in diesen Umgebungen geometrisch sehr exakt bezüglich eines Koordinatensystems beschreiben (Abbildung 6.1.1). Der große Vorteil dieser Umgebungsrepräsentation ist, dass der Roboter, je nach Auflösung des Koordinatensystems, eine sehr genaue Lokalisierung und Pfadplanung durchführen kann. Allerdings ist die exakte Erstellung einer solchen Karte wegen der Ungenauigkeit der Messung schwierig.

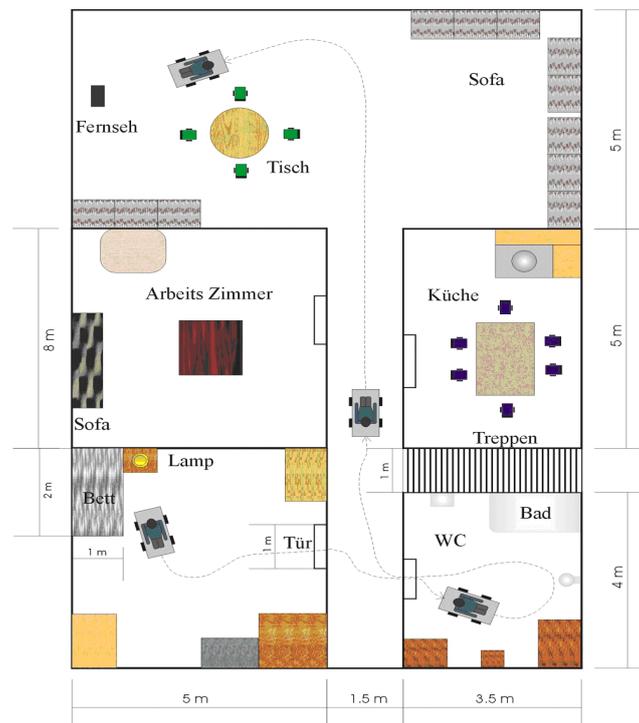


Abbildung 6.1.1: Beispiel einer geometrischen Karte ([LAU08])

b) Topologisches Modell

In einer topologischen Karte ist die Umgebung durch einen Graphen aus signifikanten Stellen (Knoten) und Transitionen (Kanten) abgebildet. Die Knoten repräsentieren markante Orte

(Bereiche) der Umgebung. Zwei Knoten sind genau dann durch eine Kante miteinander verbunden, wenn sie unmittelbar zueinander erreichbar sind. Eine topologische Repräsentation ist eine starke Vereinfachung der Umwelt, wobei keine geometrischen Verhältnisse, sondern nur Nachbarschaftsbeziehungen dargestellt werden. Kanten können Informationen (z. B. Abstand zwischen zwei Punkten, Durchgangsbreite) enthalten. Solche Karten sind sehr gut geeignet für die Pfadplanung, aber der große Nachteil liegt bei der groben Diskretisierung der Umgebung, was lediglich die Navigation zwischen, aber nicht innerhalb von Regionen erlaubt. Abbildung 6.1.2 zeigt die Umwandlung einer geometrischen Karte (z. B. die Umgebung aus Abbildung 6.1.1) in einem topologischen Modell.

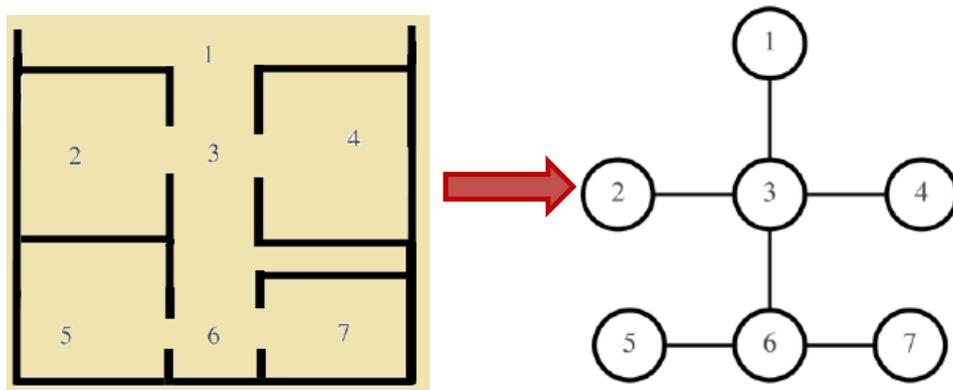


Abbildung 6.1.2: Beispiel einer topologischen Karte

c) Hybride Karte nach dem Prinzip des "analog-mapping"

Der in dieser Arbeit verwendete Ansatz basiert auf einer logischen Karte in Form eines topologischen Graphen, der aus einer geometrischen Karte durch die Verwendung des „Analog-Mapping“ Verfahrens [Bad90] gewonnen werden kann. Bei diesem Verfahren werden die Ränder der Objekte in der geometrischen Karte vergrößert, während die Ränder des Aktionsraumes verkürzt werden (Abbildung 6.1.3).

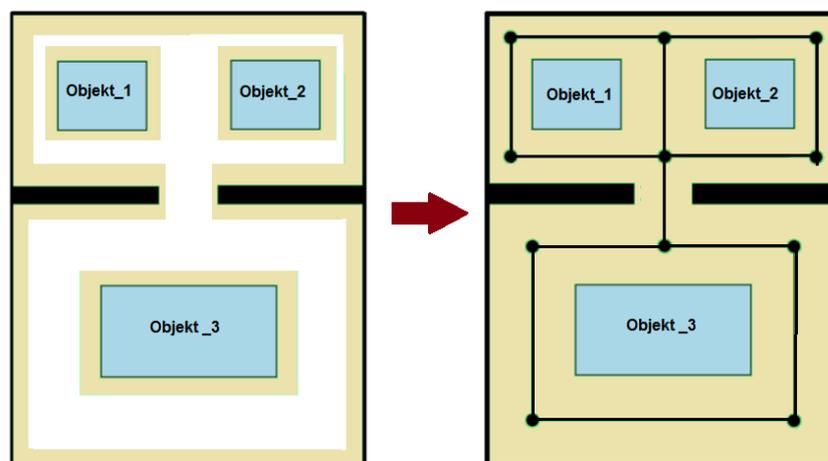


Abbildung 6.1.3: Generierung einer hybriden Karte

Man vergrößert so lange, bis die vergrößerten Objekte die Ränder anderer Objekte oder die des Aktionsraumes berühren. Man erhält einen Graphen, bestehend aus Knoten und Kanten. Die Knoten bilden Intersektionspunkte und können erweitert werden, um relevante Bereiche, wie z. B. Türdurchgangsbereiche, zu beschreiben.

6.1.2 Relative und absolute Selbstlokalisierung

Eine essentielle Eigenschaft autonomer mobiler Roboter ist die Fähigkeit zur robusten Selbstlokalisierung, d. h. zu einer möglichst genauen Schätzung der eigenen Position und Orientierung in einer Einsatzumgebung. Dabei wird zwischen einer relativen und einer absoluten Lokalisierung unterschieden. Bei der absoluten Lokalisierung wird die Position ohne Kenntnis der Ausgangsposition geschätzt, sondern durch den Einsatz von künstlichen bzw. natürlichen Landmarken ermittelt. Die relative Lokalisierung setzt eine bekannte Ausgangsposition voraus, und darauf basierend wird die nächste Position ermittelt. Ein häufig verwendetes relatives Lokalisierungsverfahren ist die so genannte Koppelnavigation („dead-reckoning“), ein mathematisches Verfahren zur Positionsbestimmung. Bei einem sich auf dem Boden bewegenden Roboter, werden Inkrementalgeber (auch Encoder genannt) an den Rädern verwendet. Mit Hilfe dieser Sensoren wird aus der Integration von Geschwindigkeiten, Winkelgeschwindigkeiten und Beschleunigungen die relative Positionsänderung des Roboters ermittelt und diese schließlich zur letzten bekannten Position addiert. Bei der absoluten Lokalisierung verwendet man häufig künstliche oder natürliche Landmarken. Bei natürlichen Landmarken handelt es sich um Gegenstände, die sich in der Umgebung befinden, wie z. B. Fenster, Tische, Wände oder Türen. Zu den künstlichen Landmarken gehören Barcodes, Lichtreflektoren, optische Markierungen oder RFID-Tags (Radio Frequency Identification), worauf Informationen über die absolute Position codiert bzw. gespeichert werden. Künstliche Landmarken bieten eine höhere Positionsgenauigkeit, da sie in bestimmten Stellen in der Umgebung platziert werden und eindeutige Informationen über die Position tragen. Der Nachteil der natürlichen Landmarken liegt in ihrer Uneindeutigkeit, wegen der Ähnlichkeit und Wiederholbarkeit in derselben Umgebung.

In dieser Arbeit wird für die relative Positionsbestimmung die Odometrie (Wegmessung) als die einfachste Form zur Realisierung der Koppelnavigation verwendet. Für die globale Positionsbestimmung wird die RFID-Technologie eingesetzt, und RFID-Landmarken werden in der Umgebung platziert.

6.1.3 Pfadplanung

Die Aufgabe des Pfadplanungsverhaltens ist die Generierung sowie die Berechnung eines kollisionsfreien und kosteneffektiven Pfades, als Abfolge der Zwischenziele von einer Start- bis zu einer Zielposition (Abbildung 6.1.4). Man unterscheidet zwischen einer lokalen und einer globalen Pfadplanung. Die globale Pfadplanung benötigt ein globales Umgebungsmodell als vereinfachte Darstellung der Umgebung. Die lokale Pfadplanung navigiert den Roboter sicher um Hindernisse herum, die plötzlich auftauchen, z. B. durch das Abtasten der Umgebung durch Ultraschallsensoren ([Cro85]). Lösungen zur globalen Pfadplanung, wie es der Fall in dieser Arbeit ist, liefern das „Analog-Mapping“ ([Bad90]) oder Voronoi-Diagramme ([Set02]-[Wag06]). Beide Verfahren unterscheiden sich dabei, wie der Start- und Zielpunkt in den logischen Graphen integriert wird. In [Bad90] werden zum Beispiel zusätzlich virtuelle Knoten im Graphen generiert, falls der Start- und Zielpunkt nicht auf einem Knoten im Graphen liegt. Ist nun der topologische Graph vorhanden, können Suchalgorithmen, wie z. B. von Dijkstra ([Dij59]-[Ale11]) oder dessen Erweiterung, der A*-Algorithmus ([Har68]-[Alk12]), zur optimalen Wegplanung genutzt werden.

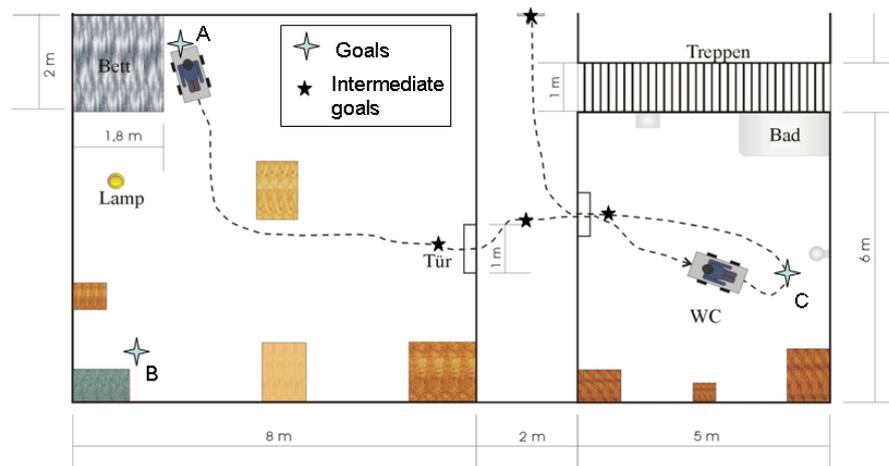


Abbildung 6.1.4: Die Pfadplanung mittels Zwischenzielen ([LAU08])

6.1.4 Die Anwendung: Autonomer Rollstuhl

Der Lehrstuhl für Automation der Universität Heidelberg hat im Rahmen eines Forschungsprojektes einen autonomen elektrischen Rollstuhl entwickelt, dessen Nutzer bei der Navigation unterstützt wird. Das System ist ein Elektrorollstuhl vom Typ OttoBock GmbH, B600, mit einem Differentialantrieb. Er besitzt zwei Castor-Räder als Fronträder, die sich passiv bewegen und nicht aktiv gesteuert werden. Die beiden Räder im Heckbereich sind von Elektromotoren angetrieben, und sie bestimmen aktiv die translatorische und rotatorische Rollstuhlbewegung. Der Rollstuhl erreicht seine drei Positionsfreiheitsgrade (x, y, φ) mittels

zwei Geschwindigkeitsfreiheitsgraden (V_x, V_ϕ), und somit handelt es sich um einen „nicht-holonomen“ Roboter. In der Praxis bedeutet diese Einschränkung, dass die Räder des Rollstuhls keine Bewegung seitwärts zulassen, sondern nur in die momentane Fahrtrichtung. Der Rollstuhl stellt ein komplexes hybrides System dar.



Abbildung 6.1.5: Ein autonomer elektrischer Rollstuhl

Um diese Komplexität des Gesamtsystems zu verringern, wurde das Steuersystem des Rollstuhls nach der RNBC-Struktur (Recursive Nested Behaviour Based Control) aufgebaut (Abbildung 6.1.6). Das gewünschte Gesamtverhalten des Rollstuhls lässt sich aus diversen Grundverhalten nach dieser verschachtelten Struktur konstruieren ([Bad94]-[Bar09]).

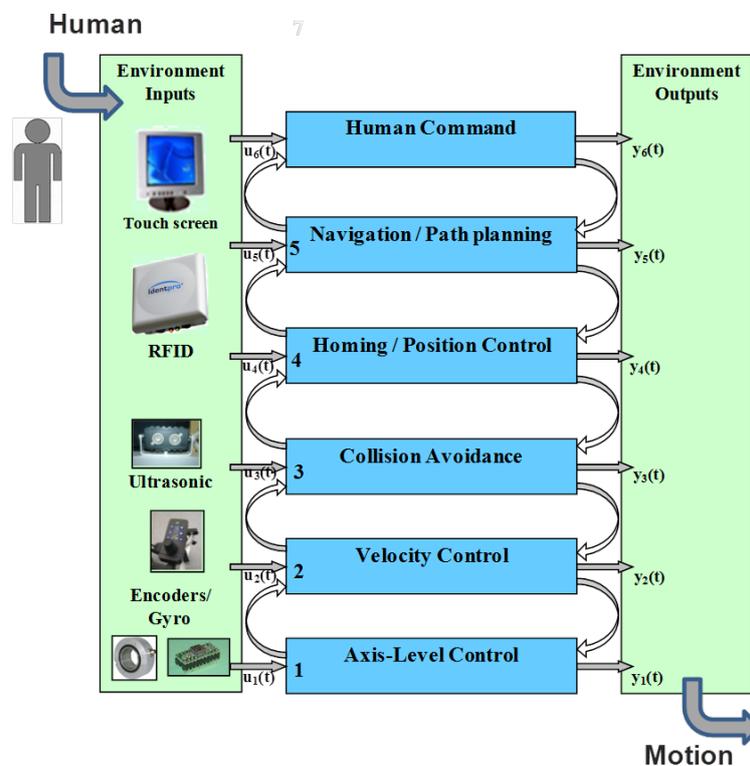


Abbildung 6.1.6: Kontrollstruktur des autonomen elektrischen Rollstuhls

Jede Verhaltensebene in diesem System übernimmt eine spezielle Aufgabe (z. B. Pfadplanung, Navigation, Kollisionsvermeidung). Der intelligente Rollstuhl ist in der Lage, autonom Zielpositionen anzusteuern und gleichzeitig Kollisionen mit Objekten zu vermeiden ([Bar03]). In dieser Arbeit wird ein Teil der Steuerungsstruktur überwacht, nämlich die drei wichtigsten Ebenen der Navigation: die Pfadplanung, Positionsregelung und Geschwindigkeitsregelung.

6.2 Entwicklung der Überwachungsstruktur

6.2.1 Das Überwachungskonzept

Über die Ebene „Human command“ gibt der Nutzer seine gewünschte Zielposition $[x, y, \varphi]_{goal}^{HC}$ über den Bildschirm ein. Diese Nutzereingabe wird an die Pfadplanungsebene (engl., Path planning) weitergegeben. Diese berechnet den optimalen Pfad von der aktuellen Position des Rollstuhls zum gewünschten Zielort. Das Ergebnis dieser Ebene ist eine geeignete Abfolge zuvor definierter Zwischenziele in der Umgebung. Beim Erreichen eines Zwischenziels wird jeweils das nächste Ziel als Vorgabe $[x, y, \varphi]_{ref}^{PP}$ an die Positionsregelungsebene (engl., Position Control) weitergegeben (Abbildung 6.2.1).

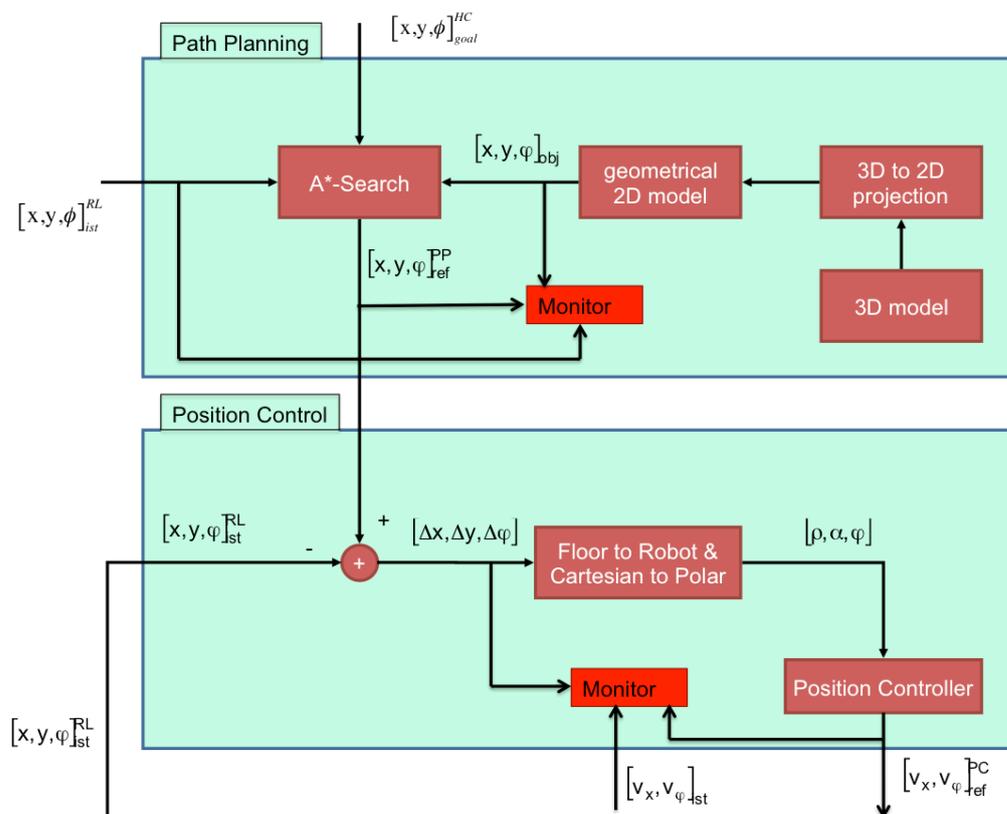


Abbildung 6.2.1: Die Verhaltensebenen Pfadplanung und Positionsregelung

Die bekannten Hindernisse werden umgangen, da diese Ebene auf a-priori-Wissen über die Positionen von Objekten $[x, y, \varphi]_{obj}$ in der gesamten Einsatzumgebung des Roboters angewiesen ist. Somit wird die Ebene der Kollisionsvermeidung (engl., Collision avoidance) für die folgenden Betrachtungen außer Acht gelassen. Der Monitor dieser Ebene soll die Ausführung der Liste der Zwischenziele überwachen, während die nächste Ebene der Positionsregelung das Erreichen der Positionen sicherstellen soll.

Die Ebene der Positionsregelung bekommt ihre Eingänge von den beiden benachbarten Ebenen: die Soll-Position $[x, y, \varphi]_{ref}^{PP}$ der Pfadplanungsebenen und die Ist-Position $[x, y, \varphi]_{ist}^{RL}$ aus der Odometrie in die Geschwindigkeitsregelungsebene (*Robot Level Control*). Aus den beiden Werten wird die Differenz $[\Delta x, \Delta y, \Delta \varphi]$ gebildet. Dieser Fehler wird mit Hilfe eines PI-Reglers (proportional–integral controller) ausgegelt. Im Anschluss findet eine Koordinatentransformation von den Welt- zu Roboterkoordinaten statt, die die Translations- und Rotationsgeschwindigkeit $[v_x, v_\varphi]_{ref}^{PC}$ an die darunter liegenden Ebenen weitergibt (Abbildung 6.2.2).

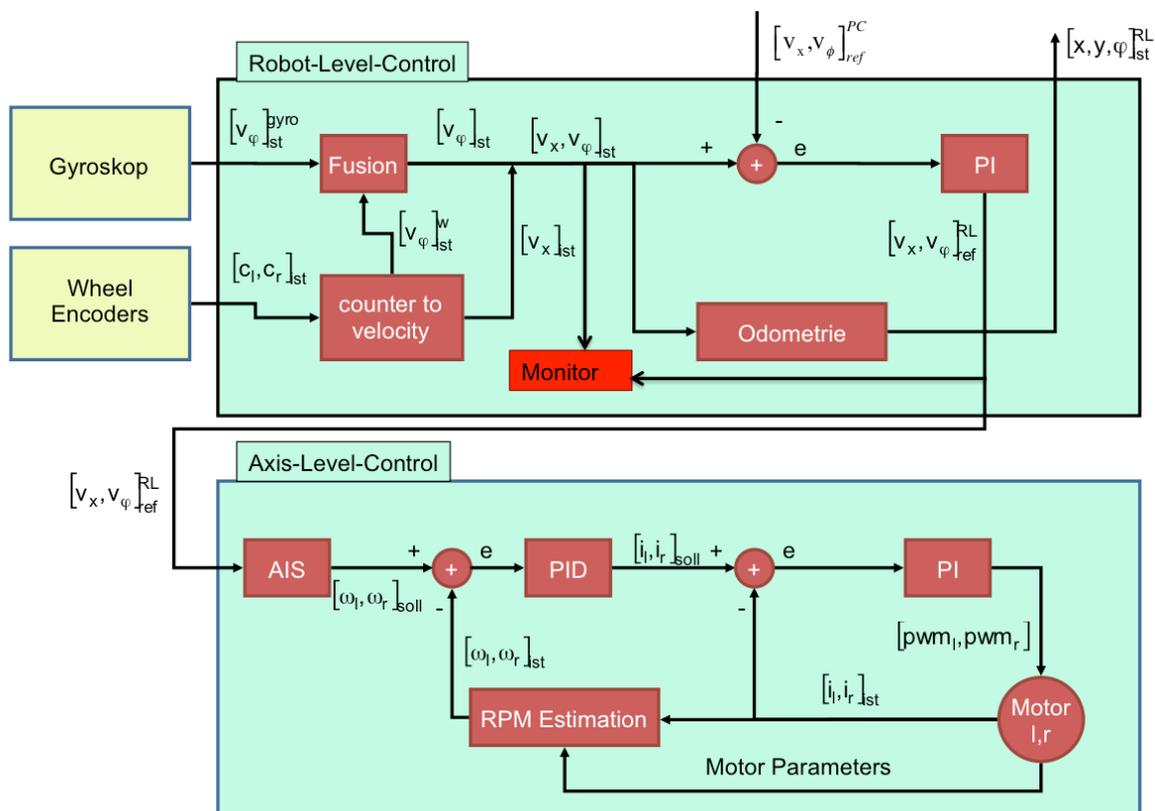


Abbildung 6.2.2: Die Verhaltensebenen der Geschwindigkeitsregelung

Die Ebene der Geschwindigkeitsregelung verwendet die Geschwindigkeitsvorgaben der übergeordneten Ebene und die Signale der Geschwindigkeitssensoren (Inkrementalgeber und

Gyroskop), um den Geschwindigkeitsregelfehler zu berechnen. Diese wird ausgeregelt und die Referenzgeschwindigkeitsvektoren werden mit Hilfe der inversen Kinematik des Rollstuhls in Referenzwerte für den Axis-Level $[V_x, V_\varphi]_{ref}^{RL}$ umgewandelt. Über eine kaskadierte Regelstruktur für die Regelung der Stromstärke, für die Geschwindigkeitsschätzung und einer Feedback-Geschwindigkeitsregelung werden die einzelnen Räder angetrieben.

6.2.2 Eingesetzte Sensorsysteme für die Überwachung

- Die Odometrie zur kontinuierlichen relativen Positionsbestimmung

Die aktuelle Position des Rollstuhls lässt sich mit Hilfe der Integration der inkrementellen Bewegungsinformation (zurückgelegte Wegstrecke, Orientierung) über die Zeit, relativ zu einem Referenzpunkt (Startposition), berechnen (Abbildung 6.2.3).

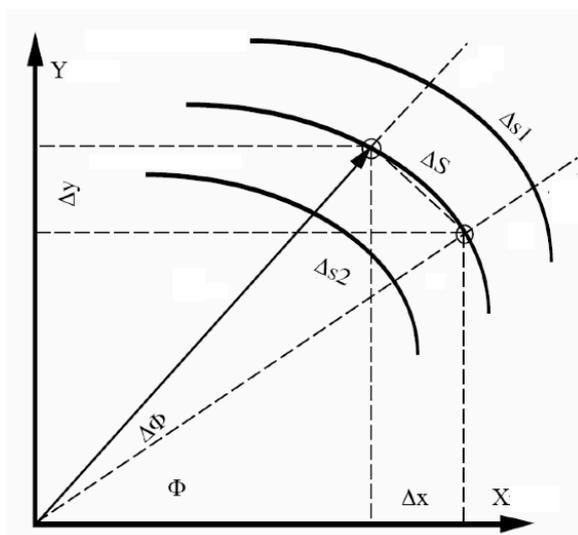


Abbildung 6.2.3: Ermittlung der zurückgelegten Strecke [Bad91]

Die zurückgelegte Strecke eines Rades wird aus dem Radius r und der Anzahl der erfassten Zählimpulse (Rotationswinkel α) des Encoders in einem Zeitintervall berechnet. Bei einem Differentialantrieb, so wie es in dieser Arbeit der Fall ist, werden die zurückgelegten Strecken ($\Delta S_1, \Delta S_2$) der einzelnen Räder (rechts und links) nach den Formeln 6.2.1 und 6.2.2 berechnet:

$$\Delta S_1 = \alpha_1 * \frac{2\pi r}{\alpha_0} \quad (6.2.1)$$

$$\Delta S_2 = \alpha_2 * \frac{2\pi r}{\alpha_0} \quad (6.2.2)$$

α_1 und α_2 sind die abgelesenen Daten aus den Encodern und α_0 ist die Anzahl der Schritte pro Umdrehung. Die gefahrene Strecke ΔS und die Orientierungsänderung $\Delta\Phi$ lassen sich aus

den zurückgelegten Strecken des rechten (Δs_1) und des linken (Δs_2) Rades sowie dem Abstand b zwischen den beiden getriebenen Rädern berechnen:

$$\Delta s = \frac{\Delta s_1 + \Delta s_2}{2} \quad (6.2.3)$$

$$\Delta \varnothing = \frac{\Delta s_1 - \Delta s_2}{b} \quad (6.2.4)$$

Die Position (x, y) sowie die Orientierung \varnothing bezüglich eines Weltkoordinatensystems ändern sich, während der Roboter fährt. Zwischen dem aktuellen und vorangegangenen Zeitpunkt bewegen sich die Räder, und man erhält die Information über die Differenz der Position. Diese Informationen über die Änderung der Position und der Richtung ermöglichen die Berechnung der aktuellen Position bzw. Orientierung des Roboters zum Zeitpunkt k aus den Werten zu einem vorangegangenen Zeitpunkt $k - 1$:

$$x_k = x_{k-1} + \Delta s \cdot \cos(\varnothing_{k-1}) \quad (6.2.5)$$

$$y_k = y_{k-1} + \Delta s \cdot \sin(\varnothing_{k-1}) \quad (6.2.6)$$

$$\varnothing_k = \varnothing_{k-1} + \Delta \varnothing \quad (6.2.7)$$

mit $k = 0, 1, 2, \dots$.

Die Odometrie ist einfach zu implementieren, aber auf längeren Strecken ist sie fehlerbehaftet und führt dazu, dass die berechnete Position nicht mehr mit der tatsächlichen übereinstimmt. Fehlerquellen der Odometrie lassen sich in systematische und nicht-systematische Fehler [BF94] einteilen. Systematische Fehler entstehen bei ungleichen Raddurchmessern, fehlerhafter Ausrichtung der Räder, begrenzter Encoderauflösung, begrenzter Encoder-Sampling-Rate oder falscher Startposition. Nicht-systematische Fehler sind Fehler, die wegen Schlupf (Verlust der Bodenhaftung) oder Unebenheiten im Boden (Bewegung über unvorhersehbare Objekte auf dem Boden) entstehen. Diese Fehler akkumulieren sich über die Zeit und führen zu einer Abweichung in der Position. Für diese Arbeit wird die Odometrie auf kurze Strecken angewendet; deswegen bietet sie eine ausreichende Genauigkeit für die Erprobung der Modifizierten Partikel-Petri-Netzen (MPPN).

- RFID-Technologie zur diskreten globalen Positionsbestimmung

Objekte bzw. Orte in der Umgebung, welche mit RFID-Tags (Abbildung 6.2.4) versehen sind, können automatisch identifiziert und lokalisiert werden. Ein RFID-Sensorsystem setzt sich

aus RFID-Transpondern (RFID-Tags) und RFID-Lesegeräten zusammen. Das Lesegerät erzeugt ein hochfrequentes elektromagnetisches Wechselfeld. Ist ein Transponder in dessen Reichweite wird der im Tag befindliche Mikrochip aktiviert und er dekodiert die Befehle des Lesegerätes.

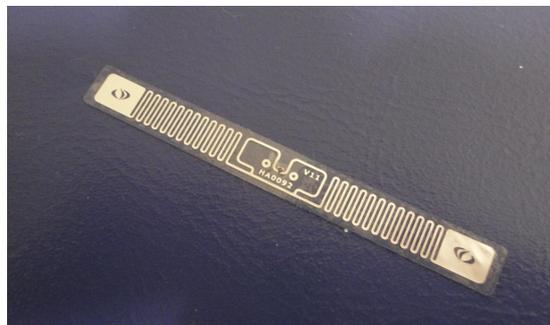
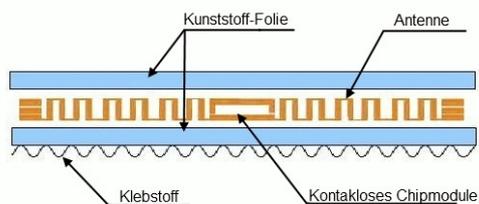


Abbildung 6.2.4: Aufbau eines RFID-Transponders

Für diese Arbeit wurde das Sensorsystem IdentMX eingesetzt. Es handelt sich um eine smarte Kombination aus Antenne und UHF-Lesegerät in einem Gehäuse. Das Kompaktlesegerät hat nur 19 cm Kantenlänge sowie eine Bautiefe von 76 mm und eignet sich für die Nutzung auch bei räumlichen Einschränkungen sowie in Fahrzeugen. Eine kombinierte Daten- und Stromversorgung erfolgt über Patch-Kabel (PoE). Das RFID-System wurde am Rollstuhl montiert, zwischen den beiden vorderen Rädern, auf der gleichen Linie wie der Mittelpunkt des Rollstuhls, so dass mit der Erkennung des Tags eine Ermittlung der Rollstuhllage erfolgt (Abbildung 6.2.5).



Abbildung 6.2.5: Anbindung des RFID-Systems am Rollstuhl

Zum RFID-System gehören grundsätzlich das Lesegerät, die RFID-Tags und eine Software zur Weiterverarbeitung der gewonnenen Informationen. Auf jedem Tag kann eine eindeutige

Information vom Typ "long" gespeichert werden, welche mit dem RFID-Lesegerät ausgelesen wird. Das Schreiben oder Ändern von Werten auf den Tags wird mit einer Software durchgeführt. Für die Rollstuhlexperimente werden auf den Tags absolute Positionen von Knoten bezüglich der Umgebungskarte gespeichert. Die Abbildung 6.2.6 zeigt das Schreiben der Position der ersten Konote in der topologischen Karte: „01“ ist die Tag-Nummer der ersten Konote auf der topologischen Karte, mit 200 als X-Wert und 630 als Y-Wert. Die Buchstaben A, B und C werden für die Trennung eingefügt.

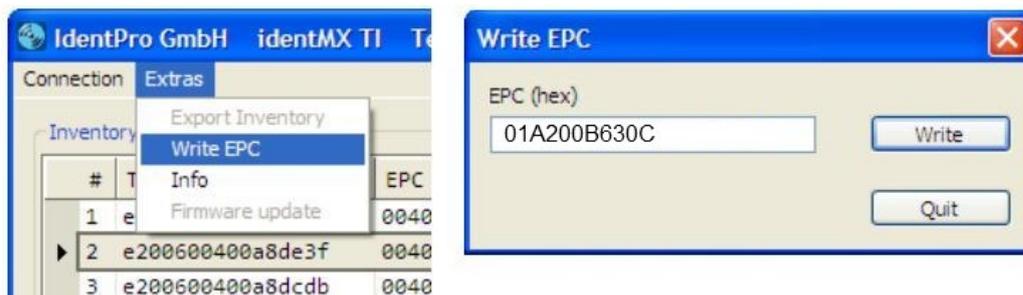


Abbildung 6.2.6: Das Schreiben der Position der ersten Knoten

Der Datenaustausch findet über eine Ethernetverbindung (TCP/IP) statt. Pakete, in Form von Header (Abbildung 6.2.7), verfolgt mit AVPs (Attribute –value Pair) (Abbildung 6.2.8), werden sowohl gesendet als auch empfangen.

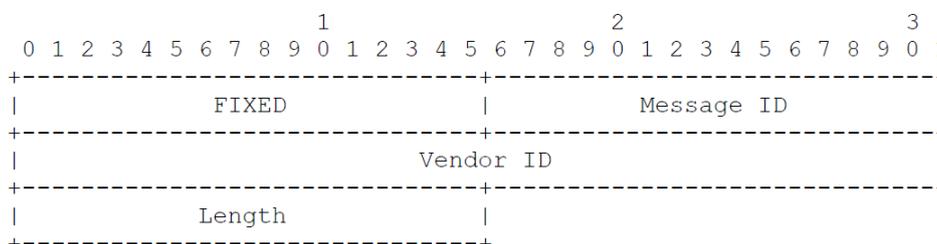


Abbildung 6.2.7: Header Format

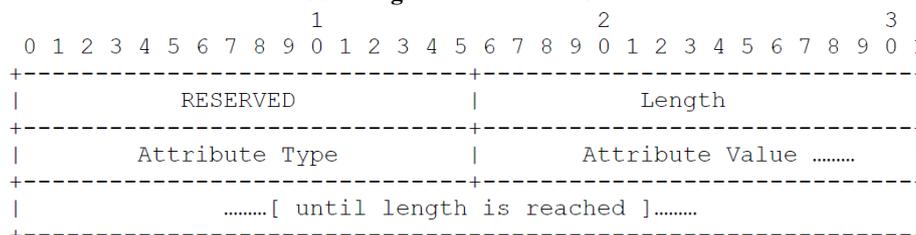


Abbildung 6.2.8: AVP Format

Das IdentMX-UHF-RFID Reader Protokoll verwendet dabei zwei logische Kommunikationskanäle: Einen Kommando-Kanal für synchrone Befehle und einen Benachrichtigungs-Kanal für asynchrone Benachrichtigungen. Alle Nachrichten (Befehle, Antworten und Benachrichtigungen) bestehen aus einem Header und einem Body. Der Header wird dekodiert, um die Art der Informationen sowie die Länge des Paketes und sein Format zu erkennen. Der Body, in der sich die Nachrichten sammeln, besteht aus einer Liste

von AVPs, die zerlegt, gelesen und sortiert werden. Unnötige AVPs werden ignoriert und aus den nötigen AVPs wird die Information extrahiert, bearbeitet und dem Benutzer in der gewünschten Form dargestellt. Zur Erfassung der Daten aus dem drahtlosen Sensorsystem (Abbildung 6.2.9) wurde in C ein Socket programmiert ([Mab12]). Dieser Socket besteht aus einer IP-Adresse zur eindeutigen Lokalisierung des Rechners und einer Port-Nummer zur Adressierung der Schnittstelle. In einer Datei sollen die aus den Tags ausgelesenen Informationen gespeichert werden.

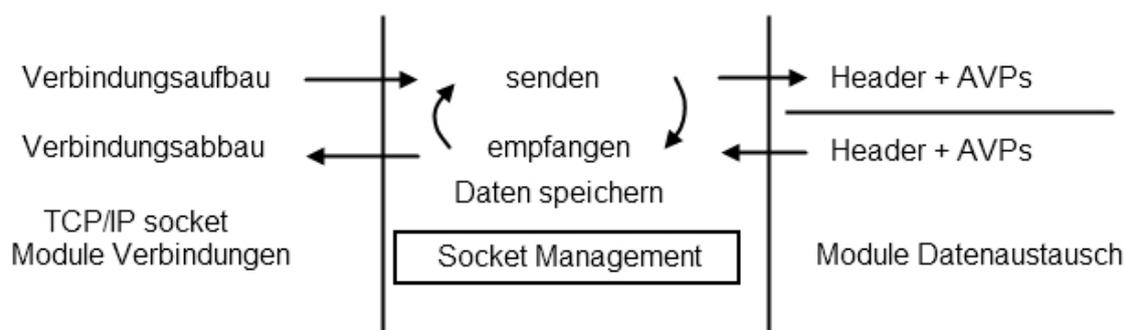


Abbildung 6.2.9: Datenaustausch im RFID-System ([Mab12])

6.2.3 Ein kombiniertes Umgebungsmodell als GUI

Ausgangspunkt einer Navigationsüberwachung ist eine geometrische Karte der Umgebung, die mit Hilfe des „Analog-Mapping“ in eine topologische Karte umgewandelt wird. Die Kombination aus topologischer und geometrischer Karte ergibt ein hybrides Umgebungsmodell. In dieser Arbeit wird von einem bekannten globalen Weltmodell ausgegangen, das in geometrischer Form vorliegt. Als Testumgebung für die Experimente dient das Roboterlabor des Lehrstuhls für Automation an der Universität Heidelberg (Abbildung 6.2.10):

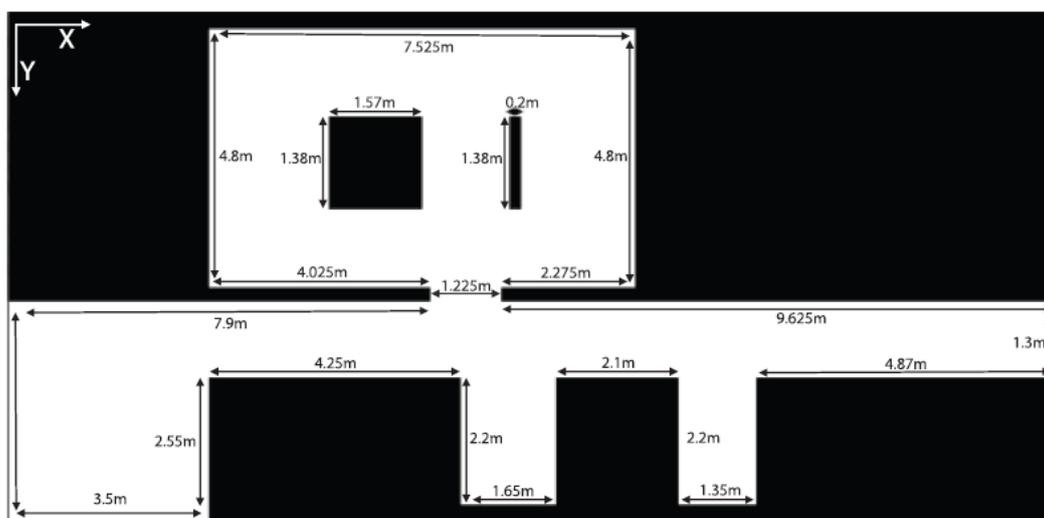


Abbildung 6.2.10: Metrische Darstellung der Testumgebung ([Ale11])

Anhand des in [Bad97] vorgeschlagenen Graphensuchalgorithmus zur Lösung des Pfadfindungsproblems, so genannte „Analog-Mapping“-Verfahren, wurde in [Ale11] die geometrische Karte in ein logisches Modell überführt. Das Ergebnis dieses Mappings stellt die topologische Karte für diese Arbeit zur Verfügung (Abbildung 6.2.11). Die Ecken der sich berührenden Objekte bilden die Knoten (grüne Punkte), und die Ränder bilden die Kanten des topologischen Graphen. An kritische Bereiche (z. B. Türen und Durchgänge) dürfen weitere Knoten (rote Punkte) definiert werden, um die Sicherheit und Genauigkeit der Navigation zu erhöhen.

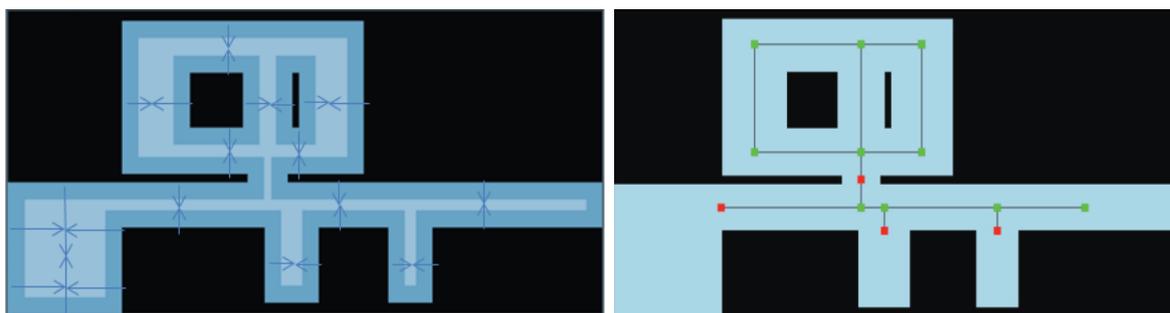


Abbildung 6.2.11: Topologische Darstellung der Testumgebung ([Ale11])

Die Knoten des topologischen Graphen haben feste Koordinaten und eine feste Gewichtung, welche den Abstand zwischen den Knoten beschreibt. Zur Pose-Referenzierung wird ein Startpunkt markiert, bei dem die Odometrie auf null gesetzt wird. Dieser Startknoten wird mit Nr.1 gekennzeichnet, und theoretisch kann jeder Punkt auf der Karte als solch ein Knoten ausgewählt werden. Auf den topologischen Knoten werden die RFID-Tags als Landmarken platziert (Abbildung 6.2.12). Auf jedem Tag sind Informationen, wie Identifikationsnummer der Knoten und ihre Position in der Umgebungskarte, gespeichert. Man beachte, dass diese Karte nur die Bereiche zeigt, welche vom Rollstuhl erreicht werden können. Aus der Korridorbreite und den geometrischen Eigenschaften des Roboters (Länge und Breite) sowie dem Wendekreis inklusive Fahrer, lässt sich bestimmen, welchen Radius der Rollstuhl bei einer Bewegung einnimmt. Die Implementierung des Umgebungsmodells wurde unter Java entwickelt und ist an die realen Gegebenheiten der Rollstuhl-Testumgebung angelehnt.

Eine für die Missionsplanung relevante Randbedingung ist der Aktionsraum des Roboters, der die Sicherheitszonen der Navigation (hellblaue Bereiche) bildet. Diesen erhält man durch die Modellierung des Roboters als Punkt. Somit werden sowohl die Ränder der geometrischen Teilkarte als auch die Hindernisse um den Radius R des Roboters erweitert (Abbildung 6.2.13).

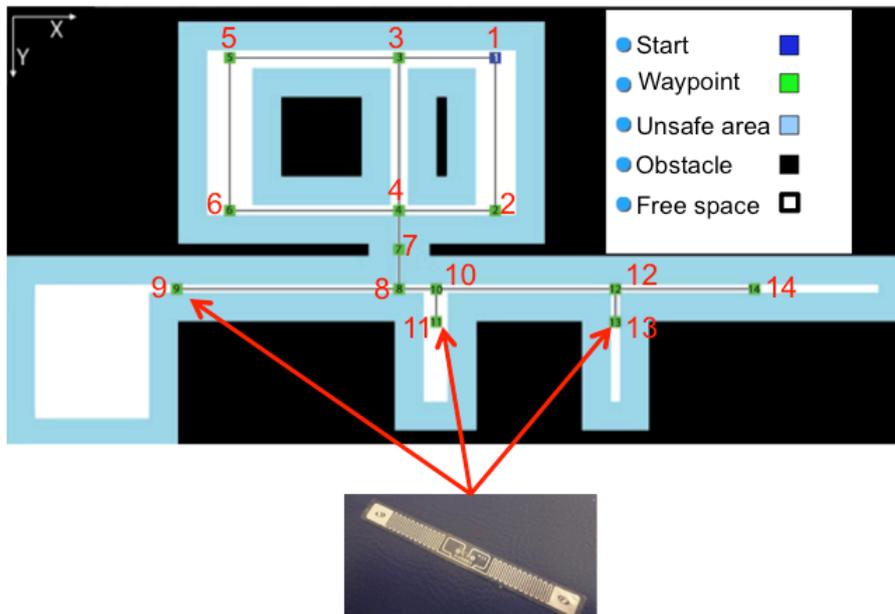


Abbildung 6.2.12: Platzierung der RFID-Tags in der Umgebung

Daraus resultiert schließlich der Freiraum F (engl., free space), der so genannte Aktionsraum, in dem jeder Punkt als Ziel für den Navigationsprozess ausgewählt werden kann. Zur Vereinfachung wurden alle Hindernisse als konvexe geometrische Körper angenommen. Mit dieser Abstraktion durch den Punkt vereinfacht sich die Routenplanung, da der Roboter nur als ein Punkt betrachtet werden muss. Darüber hinaus ist eine Adaption der Randbedingung an einen anderen Roboter, einfach durch Variation des Durchmessers möglich. Korridore, die der Roboter nicht passieren kann werden bei der Trajektorienberechnung ignoriert. Im weiteren befahrbaren Aktionsraum werden lokale Basisverhalten, wie Kollisionsvermeidung, ausgenutzt, um Hindernissen ausweichen zu können.

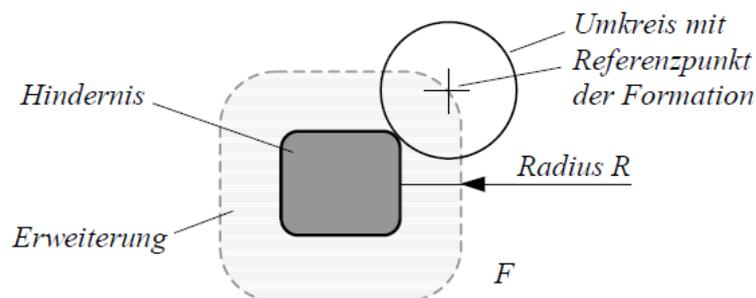


Abbildung 6.2.13: Ermittlung des Aktionsraumes

Abbildung 6.2.14 zeigt das GUI (engl., Graphical User Interface) für die Navigation. Vor der Missionsplanung werden die Randbedingungen auf das Umgebungsmodell übertragen. Beim Starten wird, basierend auf der Nutzereingabe, das Überwachungsmodell der vorliegenden Mission online generiert. Die Ausgangsposition ist mit dem blauen Quadrat gekennzeichnet

und die Zielposition mit dem roten Kreis. Der graue Kreis zeigt die aktuelle Position des Roboters. Der Referenzweg wird als rote Linie dargestellt und der geschätzte Weg als blaue Linie. Es ist möglich, weitere Landmarken in der Umgebung zu platzieren, die nicht unbedingt auf den topologischen Knoten angeordnet sein sollen. Diese Möglichkeit wird jedoch hier nicht berücksichtigt, da die Idee dieses Überwachungssystems ist, den symbolischen Teil des Überwachungsmodells direkt aus der Topologie der Landmarkenplatzierung abzuleiten. Während der Navigation wird sowohl der gewünschte Weg als auch der tatsächliche angezeigt. Statusinformationen des Roboters, wie Position, aktueller Zustand sowie Fehlermeldungen, werden dem Nutzer mitgeteilt ([Mek12a]).

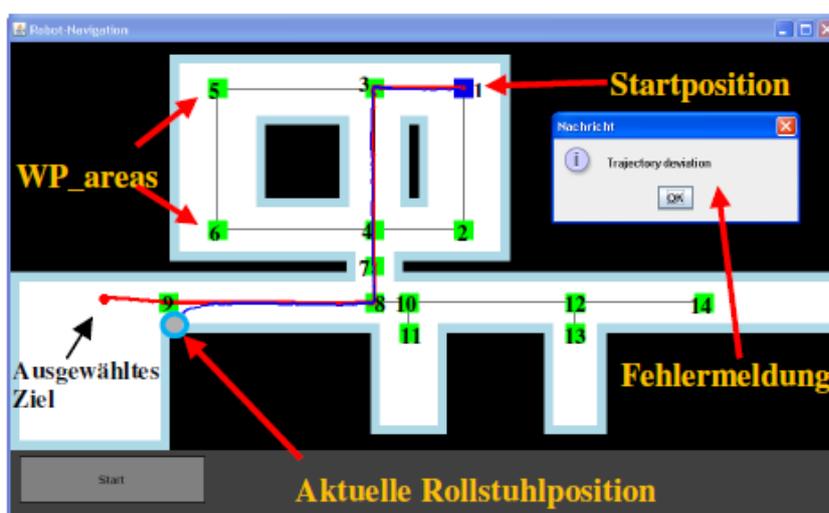


Abbildung 6.2.14: Das GUI für die Navigation ([Mek12a])

6.2.4 Definition von Weg-, Missions- und Bypasspunkten in der Steuerungsstruktur

Für eine hybride Zustandsschätzung benötigt der MPPN sowohl symbolische als auch numerische Sensorbeobachtungen. Die numerischen Beobachtungen beschreiben die Werte der kontinuierlichen Zustände des betrachteten Systems, und sie stellen am Beispiel der Rollstuhlnavigation die Odometriedaten dar. Die symbolischen Beobachtungen beschreiben Konfigurationen der Umgebung, die den Bereich um einen Knoten bzw. die Bereiche zwischen zwei Knoten darstellen. Um diese Konfiguration bestimmen zu können, werden in der realen Umgebung, genau auf den Koordinaten der Knoten der topologischen Karte, die RFID-Landmarken platziert. Auf diesen RFID-Tags werden absolute Positionen der Knoten bezüglich der Weltkarte und deren Nummer gespeichert. Das auf dem Rollstuhl montierte RFID-Lesegerät ist auf eine Reichweite von ca. 30 cm eingestellt, die gleichzeitig den Bereich um einen Knoten bestimmt und damit eine Konfiguration der Umgebung darstellt. Im

weiteren Verlauf dieser Arbeit werden die mit Landmarken gekennzeichneten Knoten als „Waypoints“ (Wegpunkte) genannt und der Detektionsbereich um einen Knoten, in dem das RFID-Lesegerät die korrespondierende Landmarke erkennen kann, wird als „Area“ bezeichnet. Nach Eingabe der Zielposition berechnet ein Trajektoriengenerator ([Ale11]) die Wegpunkte, die zur Solltrajektorie gehören. Diese bezeichnet man als „Mission Points“ (Missionspunkte), und sie bilden eine Untermenge der Wegpunkte. Eine weitere Menge von Punkten, so genannte „Bypasspoints“ (Bypasspunkte), wird definiert. Die Lage dieser Bypasspunkte bestimmt nur die Art der Bewegung zwischen den Wegpunkten, ob geradlinig oder kurvig und ob eine Position angefahren wird bzw. sich einem Objekt angenähert wird. In Abbildung 6.2.15 stellen die „Stern“-Markierungen die zu durchfahrenden Punkte dar (Bypasspunkte). Wenn es erwünscht ist, dass der Rollstuhl bestimmte Positionen in der Umgebung anfährt, werden die Bypasspunkte genau wie die Waypunkte ausgewählt. Ansonsten wird sich den Objekten nur angenähert.

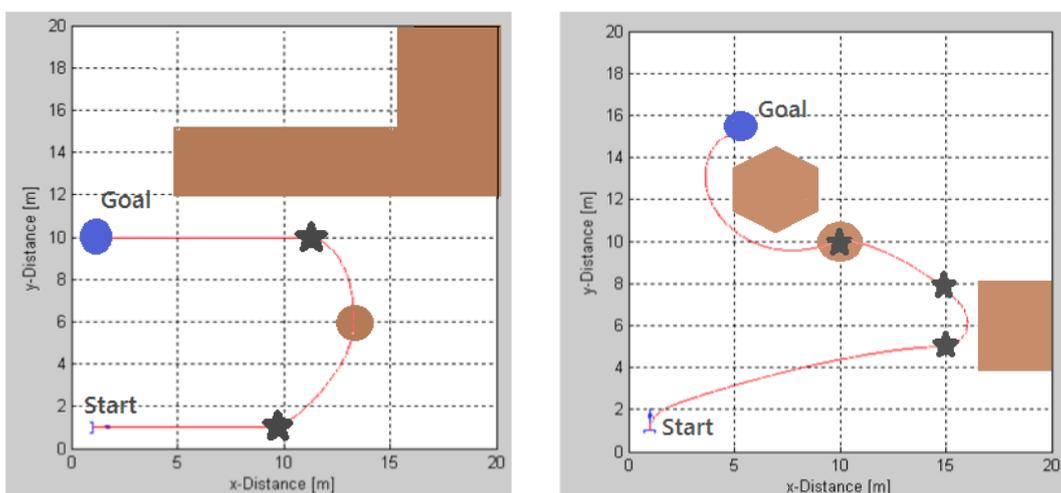


Abbildung 6.2.15: Anfahren und Annähern von Objekten mit Hilfe der Bypasspunkte

Um die Umgebung in den Code zu integrieren wurde eine Konfigurationsdatei für die topologischen Knoten erstellt, die vom Hauptprogramm der Pfadplanungsebene aufgerufen und initialisiert wird ([Alk12]). Abbildung 6.2.16 zeigt ein Beispiel für die Beschreibung von Waypoints und Bypasspoints in der Konfigurationsdatei. Sowohl Zielpositionen als auch Gegenstände, die angesteuert werden, werden als Objekte „OBJECT“ definiert. Der Parameter „size“ gibt den Radius des Objektes in Meter an. Umso geringer dieser Wert ist, desto genauer wird die Zielmarkierung angefahren. Die Variable „type“ hat den Wert 0, wenn es sich um ein Zwischenziel (sub goal) handelt und den Wert 1 für ein Endziel (final goal). Weiterhin geben „position“ und „goalPose“ die aktuelle Position (in x, y, ϕ) und die Zielposition an. Zur Berechnung der Zwischenziele und mit welchem Winkel ein Subgoal

angefahren wird, werden die Parameter „subgoalId“ und „subgoalPhi“ verwendet. Die Subgoals besitzen ebenfalls eine Position und einen Radius. Zusätzlich wird mit „edges“ der Wert der Kante des topologischen Graphen angegeben. Im Zusammenhang mit RFID-Tags-basierter Selbstlokalisierung stellen die Hauptziele die Wegpunkte „Waypoints“, an denen die Tags platziert werden, dar. Die Zwischenziele stellen die „Bypasspoints“ dar. Somit lässt sich die Karte als Konfigurationsdatei leicht beschreiben.

GOALS	SUBGOALS
OBJECT ONE type 0 position 0.0 0.0 1.0 size 0.5 goalPose 0.0 0.0 0.0 subgoalId 1 subgoalPhi 0	SUBGOAL 1 position 0.0 1.0 size 0.1 edges 2
OBJECT TWO type 1 position 0.0 2.0 1.0 size 0.1 goalPose 0.0 2.0 0.0 subgoalId 2 subgoalPhi 0	SUBGOAL 2 position 0.0 2.0 size 0.1 edges 3
OBJECT THREE type 1 position 3.25 2.0 0.0 size 0.5 goalPose 3.25 2.0 0.0 subgoalId 3 subgoalPhi 0	SUBGOAL 3 position 3.25 2.0 size 0.1 edges 4
OBJECT FOUR type 1 position 3.25 0.0 3.14 size 0.5 goalPose 3.25 0.0 3.14 subgoalId 3 subgoalPhi 0	SUBGOAL 4 position 3.25 0.0 size 0.1 edges 0

Abbildung 6.2.16: „Waypoints“ und „Bypasspoints“ als Konfigurationsdatei

6.3 Implementierung und Integration des Überwachungssystems

Auf dem Rollstuhl befindet sich ein QNX Echtzeitbetriebssystem (RTOS, engl., real-time operating system), worauf die Verhalten nach der RNBC-Struktur implementiert sind. An diesem Echtzeitrechner wurden die für die hybride Lokalisierung eingesetzten Sensorsysteme (Odometriesensoren und das RFID-Gerät) angeschlossen. Auf einem Windowsrechner befindet sich die graphische Bedienoberfläche (GUI, engl., graphical user interface) ([Ale11]) sowie eine RFID-Software, welche die Daten des RFID-Sensors zur Weiterverwendung bereitstellt ([Mab12]). Die Kommunikation zwischen diesem Host-Rechner und dem QNX-Steuerungsrechner erfolgt über eine WLAN-Bridge. Über dieser Ethernet-Schnittstelle wird mit einem Linux-System, worauf die Überwachungs- und Fehlererkennungsprozesse implementiert sind, kommuniziert. Dies beinhaltet den in Kapitel 5 beschriebenen MPPN-Simulator „PNSim“, einen Trajektorien-Generator und ein Fehlerdiagnosemodul.

Überwachung. Der BDF und die Sensordaten werden dem MPPN-Simulator „PNSim“ auf dem Linux über die beiden anderen Rechner bereitgestellt. Während der Navigation und bei auftretenden Fehlern oder ungültigen Eingaben soll dem Benutzer ein entsprechender Hinweis angezeigt und die Möglichkeit einer erneuten Eingabe angeboten werden.

6.4 Navigationsüberwachung mittels MPPN

In diesem Abschnitt werden die Modelle für die hierarchische Überwachung der Rollstuhlnavigation entworfen und erprobt. In einem vorherigen Kapitel (Abschnitt 4.6) wurde das Überwachungsmodell für die Orientierungsänderung auf der Ebene Geschwindigkeitsregelung vorgestellt. Durch eine Abfolge der Zustände "geradeaus fahren", "links abbiegen" und "rechts abbiegen" in dem Modell erreicht der Rollstuhl eine nächste Zielposition, die für die Positionsregelungsebene als Soll-Zustand definiert wird. Abbildung 6.4.1 zeigt das Überwachungsmodell dieser Ebene „Homing/Position control“, die nur das Erreichen des nächsten Zielpunktes überwacht, und zwar den des von der Pfadplanungsebene (PP) gegebenen nächsten Wegpunktes.

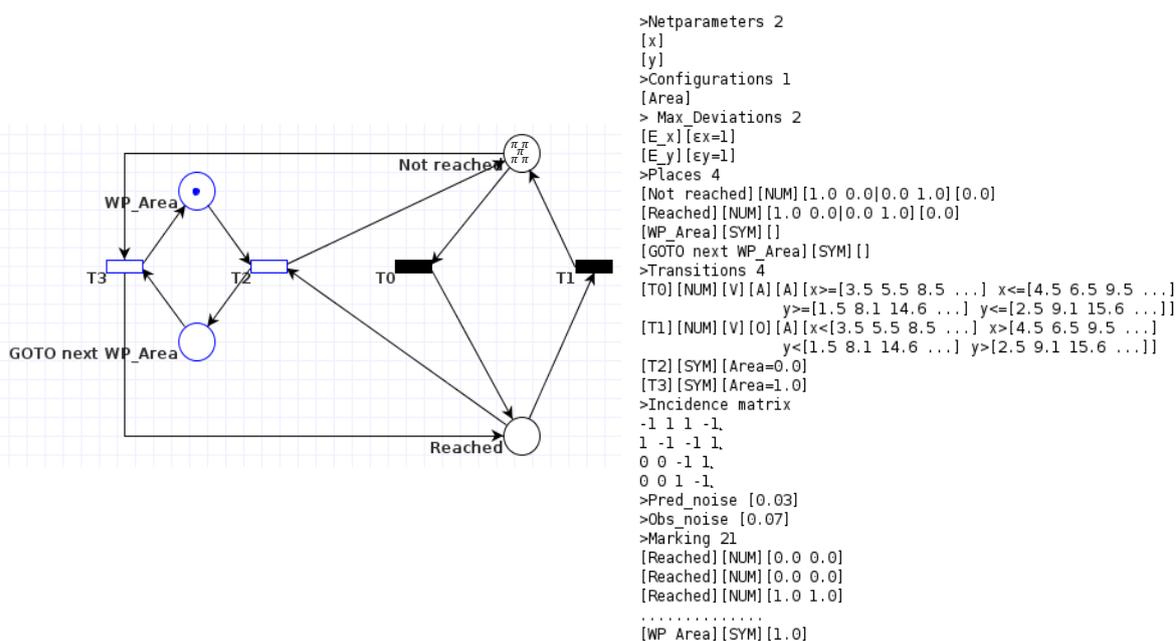


Abbildung 6.4.1: Überwachungsmodell der Positionsregelungsebene

Die Navigationsüberwachung basiert auf der Idee der probabilistischen Lokalisierung, die die Roboterposition anhand der Wahrscheinlichkeitsverteilung aller möglichen Positionen im Zustandsraum repräsentiert. In Kombination mit den Petri-Netzen stellt jeder Partikel hier nicht nur die aktuelle Position, sondern eine Hypothese über die Bereiche der Landmarken bzw. den kompletten Pfadverlauf dar. Auf diese Weise lassen sich Position und die

Landmarkenposition getrennt voneinander schätzen. Diese setzt voraus, dass die Umgebung diskretisiert wird, zum Beispiel hier durch eine topologische Karte. Als Sensoren kommen Odometrie-Sensoren und die RFID-Technologie zum Einsatz. Zur Berechnung der Wahrscheinlichkeit über alle möglichen Positionen im diskretisierten Raum wird mit dem Partikelfilter eine Teilmenge aller möglichen Positionen zur Schätzung herangenommen. Dies entspricht der Roboterposition $q_k = [x_k, y_k]$ zum Zeitpunkt k , die durch eine Menge von N gewichteten Partikeln $\pi_k^i = [q_k^i, w_k^i]$ repräsentiert wird, $i = 1 \dots N$ und w_k^i sind die Gewichte der Partikel zum Zeitpunkt k , und es gilt:

$$\sum_{i=1}^N w_k^i = 1 \quad (6.4.1)$$

Der Roboter bekommt zum Zeitpunkt $k - 1$ einen Steuerbefehl u_{k-1} als Geschwindigkeitsvorgabe der Translations- und Rotationsgeschwindigkeit $[V_x, V_\varphi]_{ref}^{PC}$, woraufhin er sich von der geschätzten Position q_{k-1} in die noch unbekannt Position q_k bewegt. Sobald der Steuerbefehl ausgeführt wird, bekommt der Roboter Sensordaten O_k , mit deren Hilfe der Roboter seine geschätzte Position bewertet. Mit dem Partikelfilter werden mehrere Positionen q_k^i verfolgt. Ist die wahre Position nahe der geschätzten, bekommt diese Position eine hohe Wahrscheinlichkeit, ansonsten eine kleine.

Durch die Anwendung der Partikel-Filterung findet die Schätzung der Roboterposition in zwei Phasen statt. In der Prädiktionsphase werden während der Roboterbewegung N neue Partikel gezogen, und sie entwickeln sich gemäß des Bewegungsmodells $f(q)$ (Abbildung 6.4.2). Somit ergibt sich die neue Roboterposition zum Zeitpunkt k :

$$q_k^i = f(q_{k-1}^i) + v_k \quad (6.4.2)$$

v_k ist das Prozessrauschen. Während der Prädiktionsphase sind alle Partikel gleichgewichtet

$$w_k^i = N^{-1} . \quad (6.4.3)$$

In der Korrekturphase werden die Partikel anhand der Sensordaten des Roboters O_k durch eine Wahrscheinlichkeitsdichtefunktion $P(q_k^i, O_k, \mu_k)$ gewichtet:

$$w_k^i = P(q_k^i, O_k, \mu_k) \quad (6.4.4)$$

μ_k ist das Beobachtungsruschen. Je näher der Partikelwert der Position der vorliegenden Messungen ist, desto größeres Gewicht bekommt er. Der Partikel mit dem höchsten Gewicht,

also mit der größten Wahrscheinlichkeit, stellt die aktuelle beste Zustandsschätzung dar. Die Entwicklung der Wahrscheinlichkeitsdichte für den Zustand ist gegeben durch:

$$p(q_k | O_k) = \eta_k p(O_k | q_k) p(q_k | O_{k-1}) \quad (6.4.5)$$

wobei η_k eine von x_k unabhängige Normalisierungskonstante ist und:

$$p(q_k | O_{k-1}) = \int_{q_{k-1}} p(q_k | q_{k-1}) p(q_{k-1} | O_{k-1}) \quad (6.4.6)$$

In einem letzten Schritt dieser Phase wird, nach der Normierung der Gewichte, ein Resampling durchgeführt Abbildung (6.4.3). In diesem Schritt werden die Partikel mit sehr kleinen Gewichtungen gelöscht, und Partikel mit den wahrscheinlichsten Hypothesen werden weiterverwendet. Dies erhöht die Genauigkeit der Schätzung und verringert den Rechenaufwand. Die restlichen ungleichgewichteten Partikel werden wieder in gleichgewichtete Partikel zerlegt. Diese stellen die Position q_{k+1}^i dar, die für die nächste Schätzung verwendet wird.

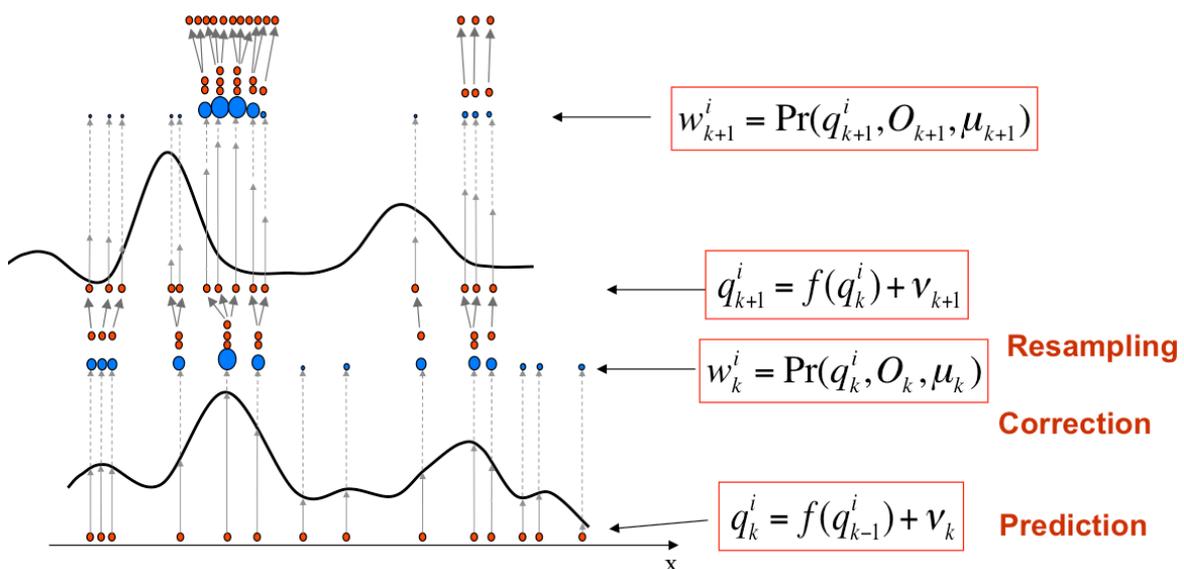


Abbildung 6.4.2: Prozess der Positionsschätzung ([Mek14])

Algorithm 9

Probabilistische Lokalisierung

Eingabe:

Partikelanzahl N ,

die Partikel zum Zeitpunkt $k-1$, $\Pi_{k-1} = \{i = 1, \dots, N | (q_{k-1}^i, w_{k-1}^i)\}$,

die Kontrolldaten u_k ,

die Position des i -ten Partikels q_k^i ,

die Sensordaten O_k

Initialisierung:

$\Pi_k = \{\}$

// Soll geresampelt werden

Falls ja

```

Für  $i=0, \dots, N$ 
  //Resampling
  Wähle ein Partikel  $i$  aus der diskreten Menge  $\Pi_{k-1}$  proportional zum Gewicht
  // Odometriemodell
   $q_k^i \sim P(q_k^i | q_{k-1}^i, u_{k-1})$ 
  // Berechne das neue Importance-Gewicht
   $w_k^i = P(O_k | q_k^i)$ 
  // Füge den Partikel in die Menge ein
   $\Pi_k = \Pi_k \cup (q_k^i, w_k^i)$ 
Sonst
  Für  $i=0, \dots, N$ 
     $q_k^i \sim P(q_k^i | q_{k-1}^i, u_{k-1})$ 
     $w_k^i = P(O_k | q_k^i)$ 
     $\Pi_k = \Pi_k \cup (q_k^i, w_k^i)$ 
Ausgabe:  $\Pi_k$ 

```

Abbildung 6.4.3: Algorithmus zur probabilistischen Lokalisierung im numerischen Modell

Durch die Gruppierung der Partikel in Klassen und die Durchführung einer Erreichbarkeitsanalyse zwischen diesen Klassen in den numerischen Stellen und den Konfigurationen in den symbolischen Stellen wird das Erreichen einer Landmarkenposition überprüft. Dieses Konzept wird auf der nächsten höheren Verhaltensebene durch die Überwachung der Pfadplanung veranschaulicht:

Während für die Ebene der Positionsregelung nur der nächste Wegpunkt zur Verfügung steht und nur das Erreichen dieses Punktes überwacht wird, wird von der Ebene „Pfadplanung“ die Ausführung der kompletten Zielliste überwacht. Mit dem Aktivieren des Rollstuhls werden sowohl das Verhalten als auch die Prozesse für das Auslesen der Odometrie- und RFID-Daten gestartet. Der Rollstuhl bewegt sich in Richtung des ausgewählten Zieles (Abbildung 6.4.4), und während der Fahrt werden die Rollstuhlposition sowie die Lage der detektierten Landmarken aufgezeichnet.

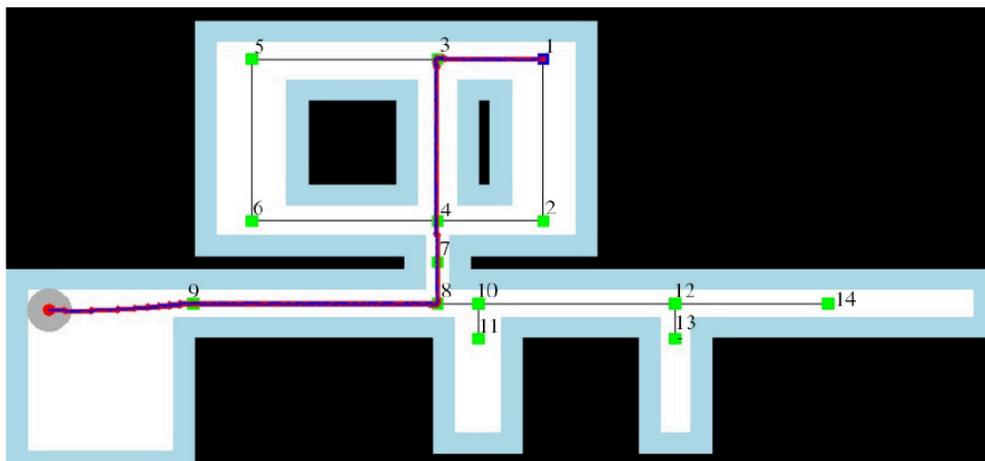


Abbildung 6.4.4: Trajektorie zum fehlerfreien Navigationszenario ([Mek14])

Für das ausgewählte Ziel in Abbildung 6.4.4 wird ein Pfad berechnet. Dieser Pfad besteht aus einer Abfolge der Knoten: 1-3-4-7-8-9. Das MPPN für dieses Experiment enthält zwei numerische Stellen, die das hybride Verhalten des Rollstuhls während der Fahrt zum Zielpunkt modellieren („Reached“ - „Not Reached“), fünf symbolische Stellen, die den topologischen Pfad bzw. die Detektion einer Area beschreiben und fünf symbolische Stellen für das Verlassen einer Area in Richtung der nächsten Area (Abbildung 6.4.5). Beide Teile des Modells sind mit hybriden Transitionen verbunden, die sowohl numerische Bedingungen in Bezug auf die Position als auch symbolische Bedingungen in Bezug auf das Erreichen bzw. Verlassen des topologischen Knoten setzen. Die Initialmarkierung des Petri-Netzes besteht aus einem symbolischen Token auf der Startknote „Area 1“ und 20 Partikeln auf dem Platz „Reached“.

$$MPPN = \langle P, T, Pre, Post, X, F_{\pi}, F_{\delta}, \Omega, M_0 \rangle$$

$$P = P^S \cup P^N \text{ mit}$$

$$P^S = \left\{ \begin{array}{l} Area1, GOTO Area 3, Area3, GOTO Area 4, Area4, GOTO Area 7, Area7, \\ GOTO Area 8, Area8, GOTO Area 9, Area9, GOTO Goal \end{array} \right\}$$

$$P^N = \{Reached, Not Reached\}; T = \{T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}\} \text{ und}$$

$$X = \{x_{pos}, y_{pos}\} \text{ mit:}$$

$$x_{pos,k+1} = x_{pos,k} + \Delta s_k \cdot \cos(\phi_{pos,k})$$

$$y_{pos,k+1} = y_{pos,k} + \Delta s_k \cdot \sin(\phi_{pos,k})$$

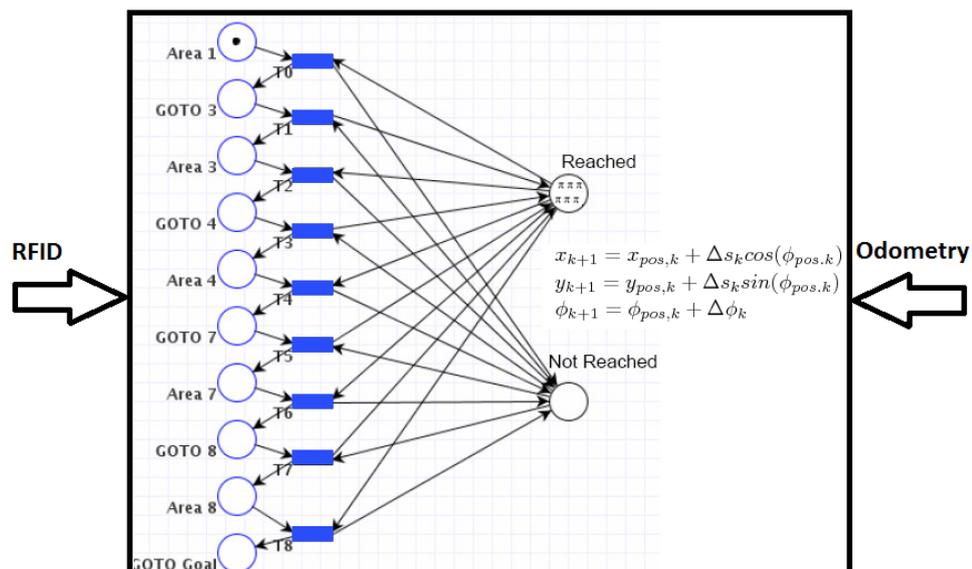


Abbildung 6.4.5: Das hybride Überwachungsmodell der Pfadplanung

Für den berechneten Pfad generiert der MPPN-Generator das entsprechende MPPN-Modell und stellt es dem Programm PNSim in Form eines BDF zur Verfügung. Dieses BDF stellt das

hybride Überwachungsmodell dar: Der diskrete Teil beschreibt das Umgebungsmodell durch die Knoten des topologischen Graphen und den Bereichen zwischen diesen Knoten und der kontinuierliche Teil stellt die kontinuierliche Dynamik des Systems dar.

Experiment 1	Trajektorie: 1-3-4-7-8-9
<pre> >Netparameters 2 [x] [y] >Configurations 1 [Area] >Max_Deviations 2 [E_x] [E_x=5] [E_y] [E_y=5] >Places 14 [Not Reached][NUM][1.0 0.0 0.0 1.0][0.0 0.0] [Reached][NUM][1.0 0.0 0.0 1.0][0.0 0.0] [Area 1][SYM][1.0] [GOTO Area 3][SYM][0.0] [Area 3][SYM][3.0] [GOTO Area 4][SYM][0.0] [Area 4][SYM][4.0] [GOTO Area 7][SYM][0.0] [Area 7][SYM][7.0] [GOTO Area 8][SYM][0.0] [Area 8][SYM][8.0] [GOTO Area 9][SYM][0.0] [Area 9][SYM][9.0] [GOTO Goal][SYM][0.0] >Transitions 11 T[0][NUM][A][402<=x<=478 40<=y<=60] T[1][NUM][A][382<=x<=402 40<=y<=60] T[2][NUM][A][382<=x<=402 60<=y<=194] T[3][NUM][A][382<=x<=402 194<=y<=214] T[4][NUM][A][382<=x<=402 214<=y<=233] T[5][NUM][A][382<=x<=402 233<=y<=253] T[6][NUM][A][382<=x<=402 253<=y<=273] T[7][NUM][A][382<=x<=402 273<=y<=293] T[8][NUM][A][180<=x<=382 273<=y<=293] T[9][NUM][A][160<=x<=180 273<=y<=293] T[10][NUM][A][87<=x<=160 273<=y<=293] >Marking 21 [Reached][NUM][488.0 50.0] [Reached][NUM][488.0 50.0] [Reached][NUM][488.0 50.0] >Pred_noise [0.8] >Obs_noise [0.9] [Area 1][SYM][1.0] </pre>	
<pre> >Forward Incidence matrix 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 </pre>	<pre> >Backward Incidence matrix 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 </pre>
>Incidence matrix	>Graph 12

1 -1 1 -1 1 -1 1 -1 1 -1 1	[0 1 1 0 0 0 0 0 0 0 0 0 0]
-1 1 -1 1 -1 1 -1 1 -1 1 -1	[1 0 0 1 0 0 0 0 0 0 0 0 0]
-1 0 0 0 0 0 0 0 0 0 0 0 0	[0 1 0 0 1 0 0 0 0 0 0 0 0]
1 -1 0 0 0 0 0 0 0 0 0 0 0	[1 0 0 0 0 1 0 0 0 0 0 0 0]
0 1 -1 0 0 0 0 0 0 0 0 0 0	[0 1 0 0 0 0 1 0 0 0 0 0 0]
0 0 1 -1 0 0 0 0 0 0 0 0 0	[1 0 0 0 0 0 0 1 0 0 0 0 0]
0 0 0 1 -1 0 0 0 0 0 0 0 0	[0 1 0 0 0 0 0 0 1 0 0 0 0]
0 0 0 0 1 -1 0 0 0 0 0 0 0	[1 0 0 0 0 0 0 0 0 1 0 0 0]
0 0 0 0 0 1 -1 0 0 0 0 0 0	[0 1 0 0 0 0 0 0 0 0 1 0 0]
0 0 0 0 0 0 1 -1 0 0 0 0 0	[1 0 0 0 0 0 0 0 0 0 0 1 0]
0 0 0 0 0 0 0 1 -1 0 0 0 0	[0 1 0 0 0 0 0 0 0 0 0 0 1]
0 0 0 0 0 0 0 0 1 -1 0 0 0	
0 0 0 0 0 0 0 0 0 1 -1 0 0	
0 0 0 0 0 0 0 0 0 0 1 -1 0	
0 0 0 0 0 0 0 0 0 0 0 1 -1	
0 0 0 0 0 0 0 0 0 0 0 0 1	

Abbildung 6.4.6: Das BDF zur Trajektorie 1-3-4-7-8-9

Das resultierende MPPN liefert in jedem Iterationsschritt eine Schätzung des hybriden Systemzustandes und die Residuen der kontinuierlichen Zustandsvariablen. Mit dem numerischen Teil des Petri-Netzes wird die Rollstuhlposition getrackt. Die numerischen Zustände "erreicht" und "nicht erreicht" (Abbildung 6.4.7) beschreiben, ob sich der Rollstuhl in seiner aktuellen Position in einer Area befindet („erreicht“ (blau)) oder nicht („nicht erreicht“ (grün)). Die Bedingungen an die numerischen Transitionen beziehen sich auf die metrische Information des Umgebungsmodells. Abbildung 6.4.7 zeigt sowohl die Schätzung des kontinuierlichen Zustandes „x-Position“ (blaue Kreuze) als auch des numerischen Zustandes „Reached / Not Reached“ (blau/grün) in Bezug auf die WP (Waypoint)_Areas (nummerierte rote Stufen). Abbildung 6.4.8 zeigt die Ergebnisse für die „y-Position“. Die ID-Nummer einer Area entspricht ihrer Höhe auf der rechten Achse (1-3-4-7-8-9).

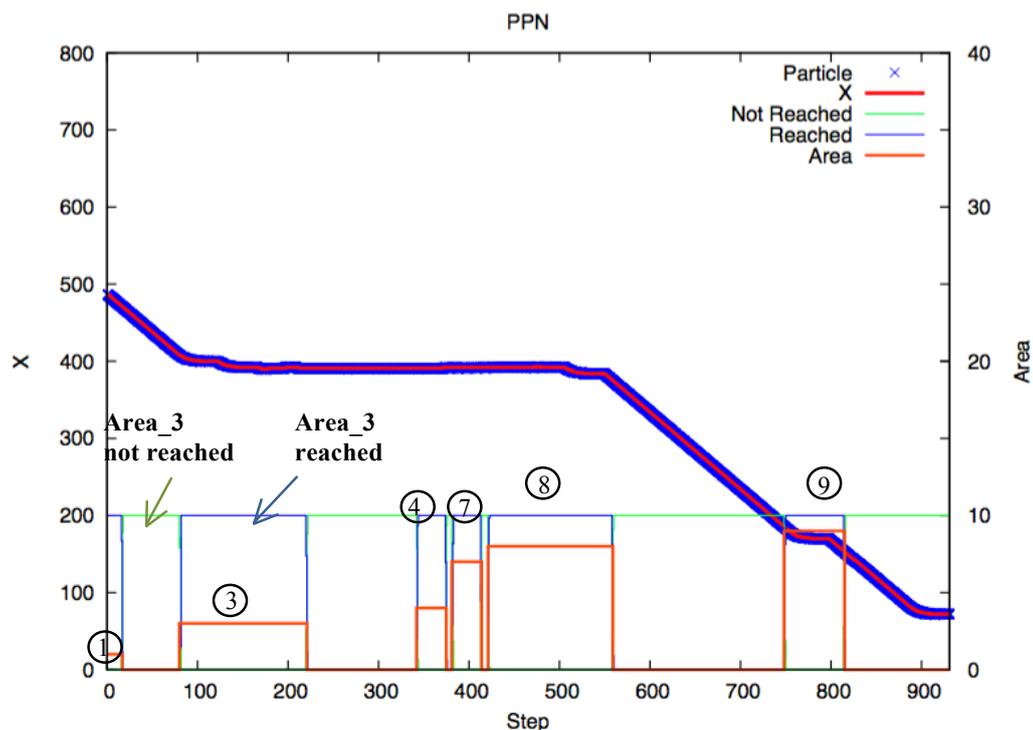


Abbildung 6.4.7: Ergebnisse der numerischen Schätzung der „x-Position“

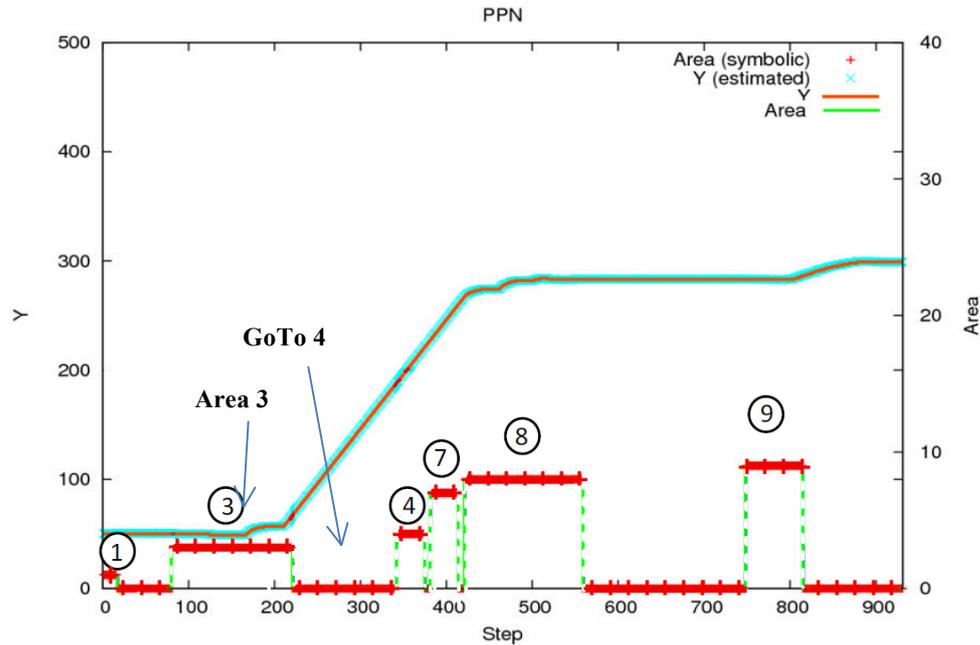


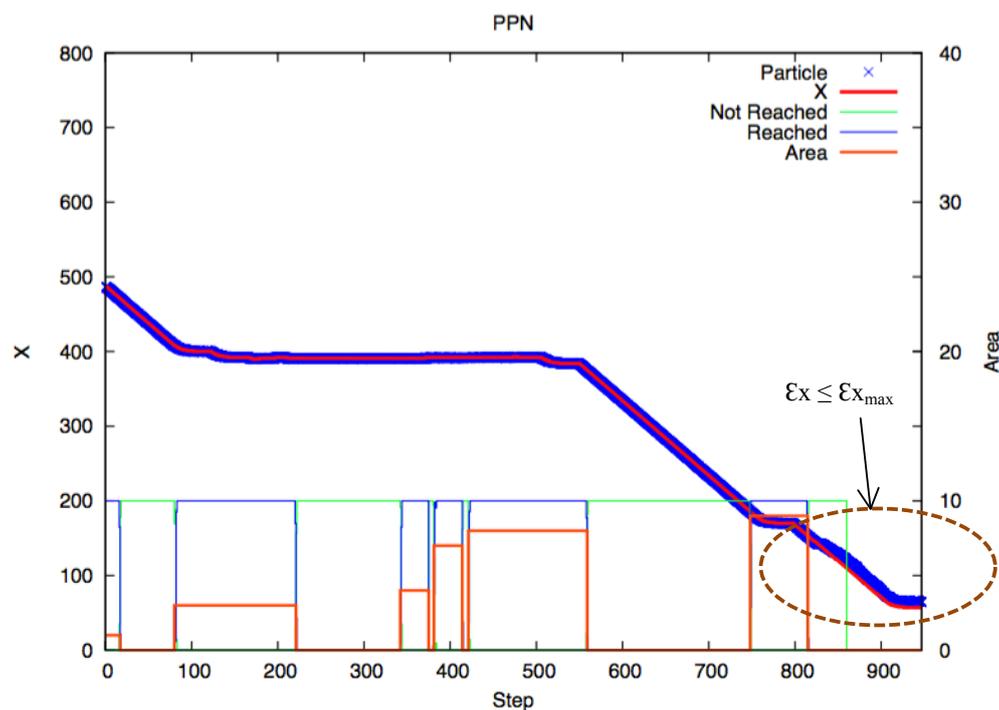
Abbildung 6.4.9: Ergebnisse der symbolischen Schätzung

Step Nr. 0	Class at Place : Reached,	Symbolic Place : Area 1	State is CONSISTENT
Step Nr. 1	Class at Place : Reached,	Symbolic Place : Area 1	State is CONSISTENT
Step Nr. 2	Class at Place : Reached,	Symbolic Place : Area 1	State is CONSISTENT
Step Nr. 14	Class at Place : Reached,	Symbolic Place : Area 1	State is CONSISTENT
Step Nr. 15	Class at Place : Reached,	Symbolic Place : Area 1	State is CONSISTENT
Step Nr. 16	Class at Place : Not Reached,	Symbolic Place : GOTO Area 3	State is CONSISTENT
Step Nr. 17	Class at Place : Not Reached,	Symbolic Place : GOTO Area 3	State is CONSISTENT
Step Nr. 18	Class at Place : Not Reached,	Symbolic Place : GOTO Area 3	State is CONSISTENT
Step Nr. 79	Class at Place : Not Reached,	Symbolic Place : GOTO Area 3	State is CONSISTENT
Step Nr. 80	Class at Place : Not Reached,	Symbolic Place : GOTO Area 3	State is CONSISTENT
Step Nr. 81	Class at Place : Reached,	Symbolic Place : Area 3	State is CONSISTENT
Step Nr. 82	Class at Place : Reached,	Symbolic Place : Area 3	State is CONSISTENT
Step Nr. 218	Class at Place : Reached,	Symbolic Place : Area 3	State is CONSISTENT
Step Nr. 219	Class at Place : Reached,	Symbolic Place : Area 3	State is CONSISTENT
Step Nr. 220	Class at Place : Not Reached,	Symbolic Place : GOTO Area 4	State is CONSISTENT
Step Nr. 221	Class at Place : Not Reached,	Symbolic Place : GOTO Area 4	State is CONSISTENT
Step Nr. 222	Class at Place : Not Reached,	Symbolic Place : GOTO Area 4	State is CONSISTENT
Step Nr. 340	Class at Place : Not Reached,	Symbolic Place : GOTO Area 4	State is CONSISTENT
Step Nr. 341	Class at Place : Not Reached,	Symbolic Place : GOTO Area 4	State is CONSISTENT
Step Nr. 342	Class at Place : Reached,	Symbolic Place : Area 4	State is CONSISTENT
Step Nr. 343	Class at Place : Reached,	Symbolic Place : Area 4	State is CONSISTENT
Step Nr. 372	Class at Place : Reached,	Symbolic Place : Area 4	State is CONSISTENT
Step Nr. 373	Class at Place : Reached,	Symbolic Place : Area 4	State is CONSISTENT
Step Nr. 374	Class at Place : Not Reached,	Symbolic Place : GOTO Area 7	State is CONSISTENT
Step Nr. 375	Class at Place : Not Reached,	Symbolic Place : GOTO Area 7	State is CONSISTENT
Step Nr. 379	Class at Place : Not Reached,	Symbolic Place : GOTO Area 7	State is CONSISTENT
Step Nr. 380	Class at Place : Not Reached,	Symbolic Place : GOTO Area 7	State is CONSISTENT
Step Nr. 381	Class at Place : Reached,	Symbolic Place : Area 7	State is CONSISTENT
Step Nr. 382	Class at Place : Reached,	Symbolic Place : Area 7	State is CONSISTENT
Step Nr. 411	Class at Place : Reached,	Symbolic Place : Area 7	State is CONSISTENT
Step Nr. 412	Class at Place : Not Reached,	Symbolic Place : Area 7	State is CONSISTENT
Step Nr. 413	Class at Place : Not Reached,	Symbolic Place : GOTO Area 8	State is CONSISTENT
Step Nr. 414	Class at Place : Not Reached,	Symbolic Place : GOTO Area 8	State is CONSISTENT
Step Nr. 419	Class at Place : Not Reached,	Symbolic Place : GOTO Area 8	State is CONSISTENT
Step Nr. 420	Class at Place : Not Reached,	Symbolic Place : GOTO Area 8	State is CONSISTENT
Step Nr. 421	Class at Place : Reached,	Symbolic Place : Area 8	State is CONSISTENT
Step Nr. 422	Class at Place : Reached,	Symbolic Place : Area 8	State is CONSISTENT
Step Nr. 556	Class at Place : Reached,	Symbolic Place : Area 8	State is CONSISTENT

Step Nr. 557	Class at Place : Reached,	Symbolic Place : Area 8	State is CONSISTENT
Step Nr. 558	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 559	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 746	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 747	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 748	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 749	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 812	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 813	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 814	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 815	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 927	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 928	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 929	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 930	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT

Abbildung 6.4.10: Ergebnisse der Konsistenzanalyse

Mit der Zeit könnten sich Odometriefehler akkumulieren und zu einer Abweichung von der Soll-Trajektorie führen. Liegt diese Abweichung ε innerhalb des maximalen zulässigen Bereichs ε_{\max} (aus der Spezifikation), wird diese vom Diagnosemodul nicht als Fehler gemeldet. Der Systemzustand bleibt somit konsistent (Abbildung 6.4.11).



Step Nr. 808	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 809	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 810	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 811	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 812	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 813	Class at Place : Reached,	Symbolic Place : Area 9	State is CONSISTENT
Step Nr. 814	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 815	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 816	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 817	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 818	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 819	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 820	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 821	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 822	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT

Step Nr. 823	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 926	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 927	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 928	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 929	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 935	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 936	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 937	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 938	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 939	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT
Step Nr. 940	Class at Place : Not Reached,	Symbolic Place : GOTO Goal	State is CONSISTENT

Abbildung 6.4.11: Überwachungsergebnisse bei Abweichungen innerhalb des spezifizierten Bereichs

In einem weiteren Experiment wurde das Ziel im ersten engen Korridor ausgewählt. Hierfür ergibt sich ein neuer Pfad: 1-3-4-7-8-10-11. Die Abbildung 6.4.12 zeigt die Ergebnisse des numerischen Modells und Abbildung 6.4.13 zeigt die Ergebnisse des symbolischen Modells.

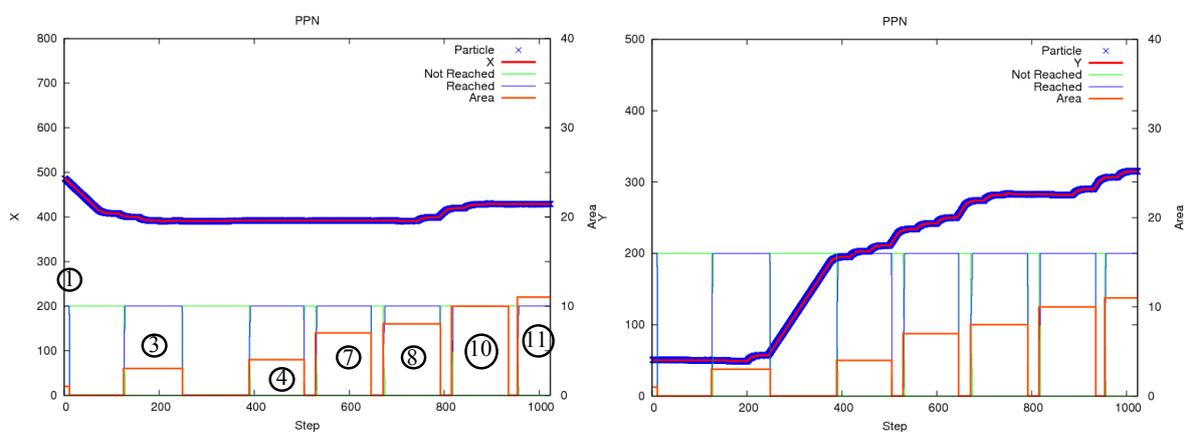


Abbildung 6.4.12: Ergebnisse der numerischen Schätzung für die x-Position und die y-Position

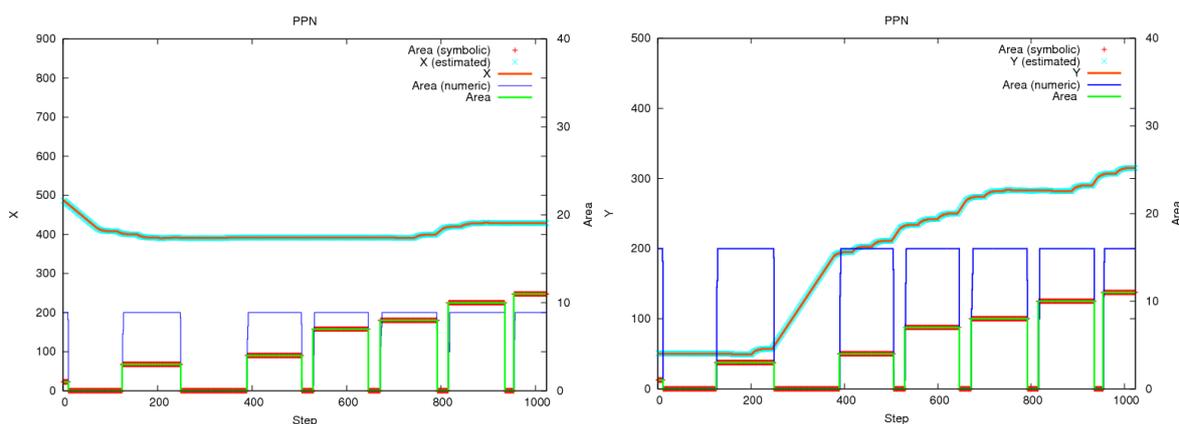


Abbildung 6.4.13: Ergebnisse der symbolischen Schätzung für die x-Position und die y-Position

7 Evaluierung des Verfahrens

Das Verfahren wird in diesem Kapitel auf verschiedene Kriterien, wie z. B. die Empfindlichkeit, die Robustheit und die Dimension, evaluiert. Zum Testen der Robustheit und Empfindlichkeit des Fehlerdiagnosemechanismus in Abschnitt 7.1 werden Fehlerszenarien durch eine Fehlerinjektion generiert und betrachtet. In Abschnitt 7.2 wird der Vorteil des Verfahrens, in Bezug auf die Reduktion der Komplexität durch die Verwendung von Unterräumen sowie durch die hierarchische Struktur des Überwachungskonzeptes, erläutert. Die Evaluation der Nutzbarkeit des Verfahrens wird in Abschnitt 7.3 durch die Integration des Verfahrens in weiteren Anwendungen gezeigt.

7.1 Robustheit und Empfindlichkeit

Um die Robustheit und die Empfindlichkeit des Überwachungs- und Fehlerdiagnosemechanismus in Bezug auf die Detektion von Inkonsistenzen zu untersuchen, werden drei Fehlerklassen definiert:

- Inkonsistenzen im Systemverhalten als eine unzulässige Abweichung vom Normalzustand,
- eine kurzzeitige Störung aufgrund nicht kontrollierbarer äußerer Einflüsse,
- ein interner Komponentenfehler, der zur Unterbrechung der Systemfunktion führt.

Durch Fehlerinjektionsszenarien in den beiden Anwendungsbeispielen, die Temperaturregelung und die Roboternavigation, wird das Verfahren auf die verschiedenen Fehlerklassen hin untersucht.

- a) Evaluation des Navigationsbeispiels

Inkonsistenzen im Navigationsverhalten: Abbildung 7.1.1 zeigt den Fall, dass der Rollstuhl von der geplanten Fahrtrajektorie abweicht, und somit wird der Tag Nr. 9 nicht detektiert.

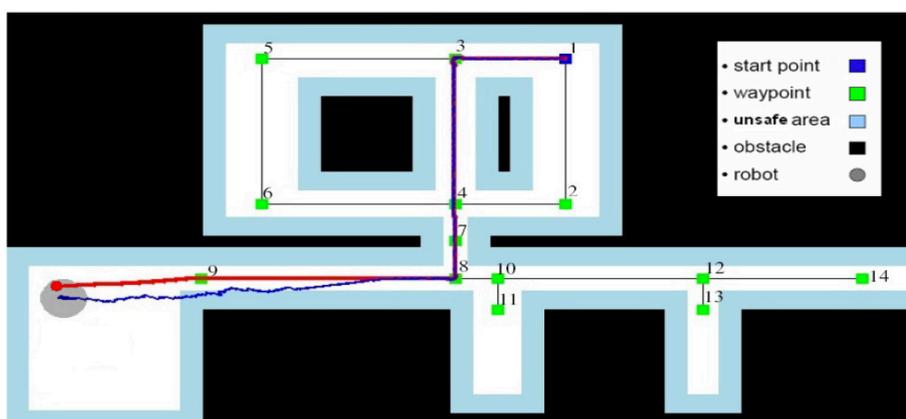


Abbildung 7.1.1: Experiment zum inkonsistenten Navigationsverhalten ([Mek14])

Die Ergebnisse der numerischen sowie der symbolischen Schätzung sind in Abbildung 7.1.2 dargestellt. Die Konsistenzanalyse des symbolischen Zustands in Bezug auf den numerischen zeigt in Abbildung 7.1.3 einen Übergang von einem konsistenten zu einem inkonsistenten Zustand (Zeitschritt Nr. 778).

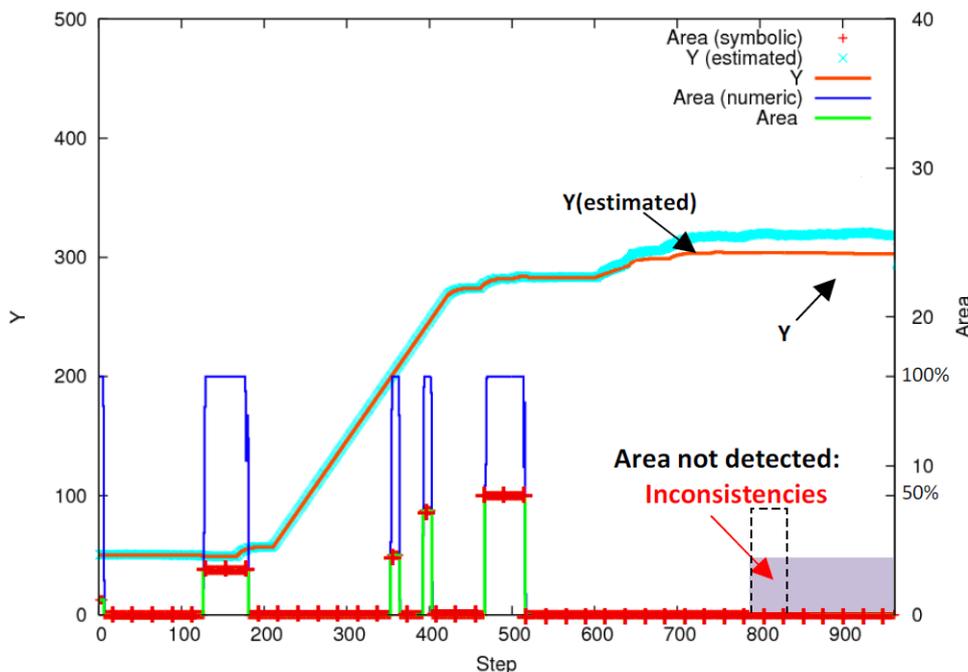


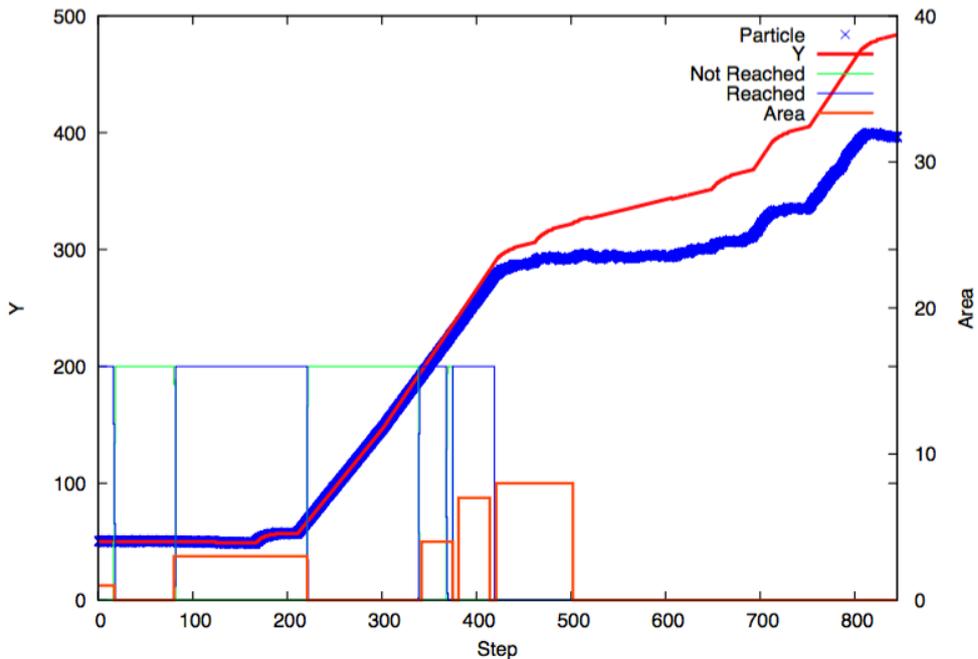
Abbildung 7.1.2: Detektion von Inkonsistenzen

Step Nr. 772	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 773	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 774	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 775	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 776	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 777	Class at Place : Not Reached,	Symbolic Place : GOTO Area 9	State is CONSISTENT
Step Nr. 778	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 779	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 780	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 781	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 782	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 783	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 784	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 785	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 786	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 787	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 788	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
...			
Step Nr. 828	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 829	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 830	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 831	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 832	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 833	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 834	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 835	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 836	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
Step Nr. 837	Class at Place : Reached,	Symbolic Place : GOTO Area 9	State is INCONSISTENT
...			

Abbildung 7.1.3: Ergebnisse der Konsistenzanalyse

Die Abweichung der kontinuierlichen Größe Y in Abbildung 7.1.2 ist nicht der Auslöser der Inkonsistenz, weil sich dieser im Interwall $[\epsilon_{max} \epsilon_{safe}]$ befindet. Denn die Überschreitung von ϵ_{max} wird vom Diagnosemodul nur als Störung klassifiziert. Sollte aus den anderen Ebenen keine Rückmeldung über das Auftreten von Fehlern geschickt werden (z. B Problem

an den Rädern) oder es wird keine Reaktion auf die unerwartete Situation ausgelöst (z. B. Abweichung von der Soll-Trajektorie, um ein unbekanntes Hindernis zu vermeiden), wird keine Inkonsistenz gemeldet. Abbildung 7.1.4 zeigt den Fall einer Überschreitung von ε_{safe} , was zu einer Inkonsistenz im Systemverhalten führt.



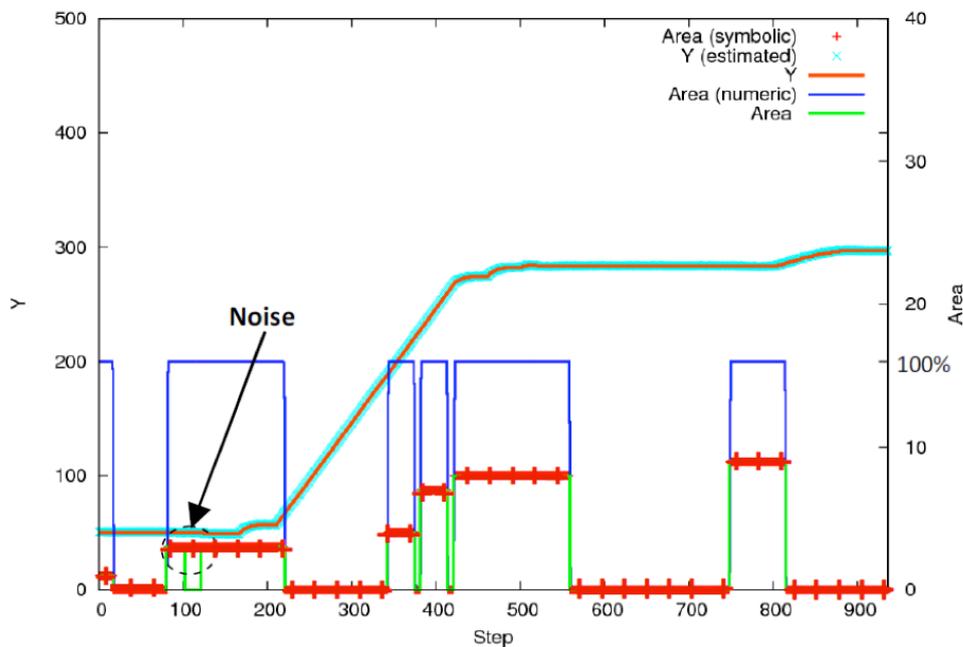
Step Nr. 508	Class at Place : Reached,	Symbolic Place : Area 4State is CONSISTENT
Step Nr. 509	Class at Place : Reached,	Symbolic Place : Area 4State is CONSISTENT
Step Nr. 510	Class at Place : Reached,	Symbolic Place : Area 4State is CONSISTENT
Step Nr. 511	Class at Place : Reached,	Symbolic Place : Area 4State is CONSISTENT
Step Nr. 512	Class at Place : Reached,	Symbolic Place : Area 4State is CONSISTENT
Step Nr. 513	Class at Place : Reached,	Symbolic Place : Area 4State is CONSISTENT
Step Nr. 514	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 515	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 516	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 517	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 518	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 519	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 520	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 521	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
...		
Step Nr. 649	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 650	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 651	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 652	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 653	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 654	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
...		
Step Nr. 839	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 840	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 841	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 842	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 843	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT
Step Nr. 844	Class at Place : Not Reached,	Symbolic Place : Area 4State is INCONSISTENT

Abbildung 7.1.4: Experiment zur Überschreitung der Sicherheitsgrenze

Eine kurzzeitige Störung der Beobachtung:

Eine Störung könnte im kontinuierlichen Teil durch die Überschreitung von ε_{max} sowie auch im symbolischen Teil durch eine kurzzeitige Unerreichbarkeit eines symbolischen Zustands

auftreten. Abbildung 7.1.5 zeigt den Fall einer Störung in der symbolischen Beobachtung, und zwar während des Eintretens in ein Area (Zeitschritte 104 bis 116). Dieses Szenario tritt beispielsweise auf, wenn sich ein metallischer Gegenstand auf dem Boden befindet und somit die Detektion eines RFID-Tags kurz stört. An dieser Stelle und Dank der Betrachtung der Wechselwirkungen zwischen beiden Teilen des Modells, symbolisch und numerisch, klassifiziert das Diagnosemodul diese Abweichung als Störung, und der Systemzustand wird weiterhin als konsistent angezeigt. Denn trotz der fehlenden symbolischen Beobachtung weisen die kontinuierlichen Variablen (x- und y-Positionen) keine Abweichungen auf. Somit werden unnötige Fehlermeldungen, die wiederum zum Abbrechen des Navigationsprozesses führen, nicht ausgelöst.

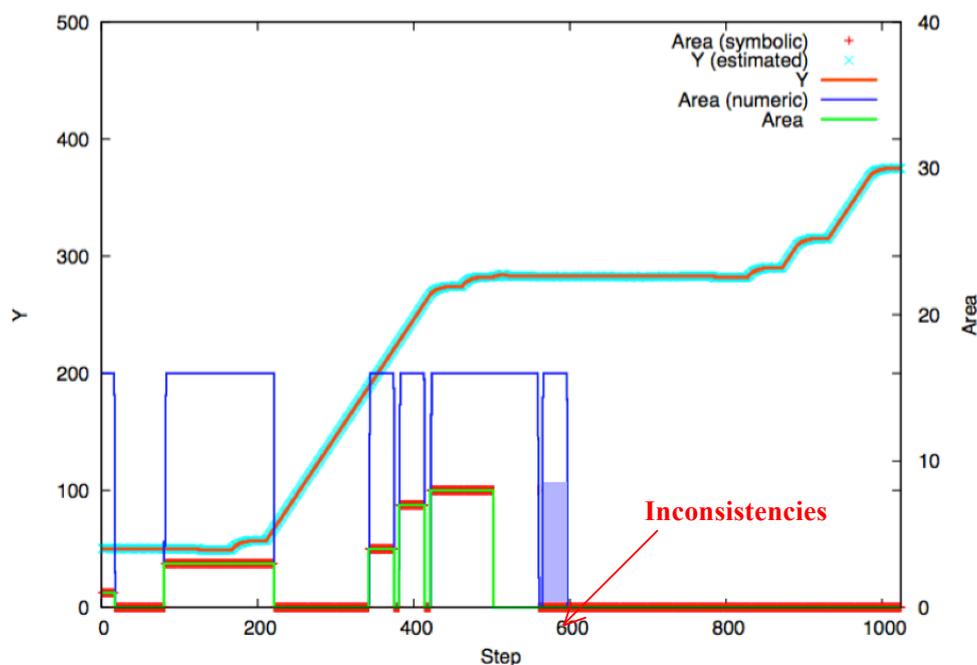


Step Nr. 97	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 98	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 99	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 100	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 101	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 102	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 103	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 104	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 105	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 106	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 107	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 108	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 109	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 110	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 111	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 112	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 113	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 114	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT
Step Nr. 115	Class at Place : Reached,	Symbolic Place : Area 3State is CONSISTENT

Abbildung 7.1.5: Experiment zur kurzzeitigen Störung

Ausfall des RFID-Systems:

Abbildung 7.1.6 zeigt das Szenario zur Abwesenheit der symbolischen Messung während der Durchfahrt über ein Area, gefolgt von einem totalen Ausfall des RFID-Systems. Die Abwesenheit der Beobachtung wurde in einer ersten Phase vom Diagnosemodul als klassische Störung detektiert. Die Nichtdetektion des Zustandswechsels beim Verlassen des Area und Eintreten des nächsten Area führt zum Auslösen einer Inkonsistenzmeldung.



Step Nr. 562	Class at Place : Not Reached,	Symbolic Place : GOTO Area 10	State is CONSISTENT
Step Nr. 563	Class at Place : Not Reached,	Symbolic Place : GOTO Area 10	State is CONSISTENT
Step Nr. 564	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 565	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 566	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 567	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 568	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 569	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 570	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 591	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 592	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 593	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT
Step Nr. 594	Class at Place : Reached,	Symbolic Place : GOTO Area 10	State is INCONSISTENT

Abbildung 7.1.6: Experiment zum Komponentenfeler

Dieses Szenario sah am Anfang dem Fall einer kurzzeitigen Störung ähnlich, aber dank der Erreichbarkeitsanalyse war es möglich, zwischen einer externen Störung und einem internen Komponentenfeler zu unterscheiden.

b) Evaluation des Temperaturbeispiels

Die Empfindlichkeit des Verfahrens am Temperaturbeispiel wird durch eine prompte Änderung im Temperaturverhalten evaluiert. Eine Fehlerinjektion (Nach einem kurzzeitigen Temperaturanstieg das System plötzlich ausschalten) im Experiment in Abbildung 7.1.7.a

führt zu einem schnellen Wechsel zwischen den Modi „Heizen“ und „Kühlen“ in Abbildung 7.1.7.b. Der Überwacher hat den Zustandswechsel richtig detektiert. Der selektierte Modus Cooling_Off wurde als höchstwahrscheinlicher Zustand geschätzt.

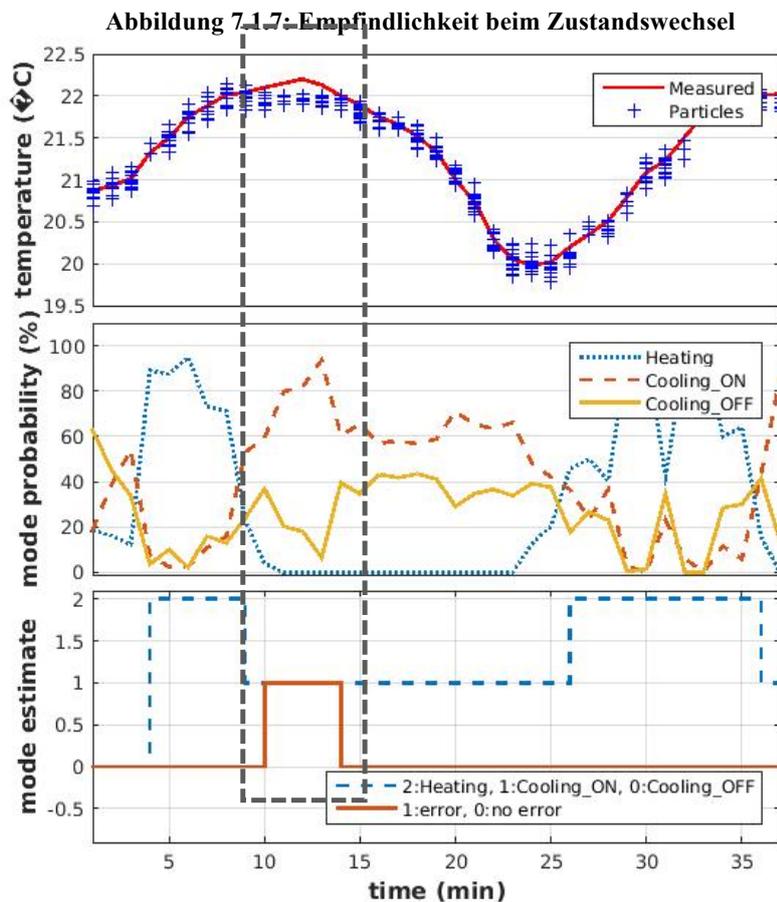
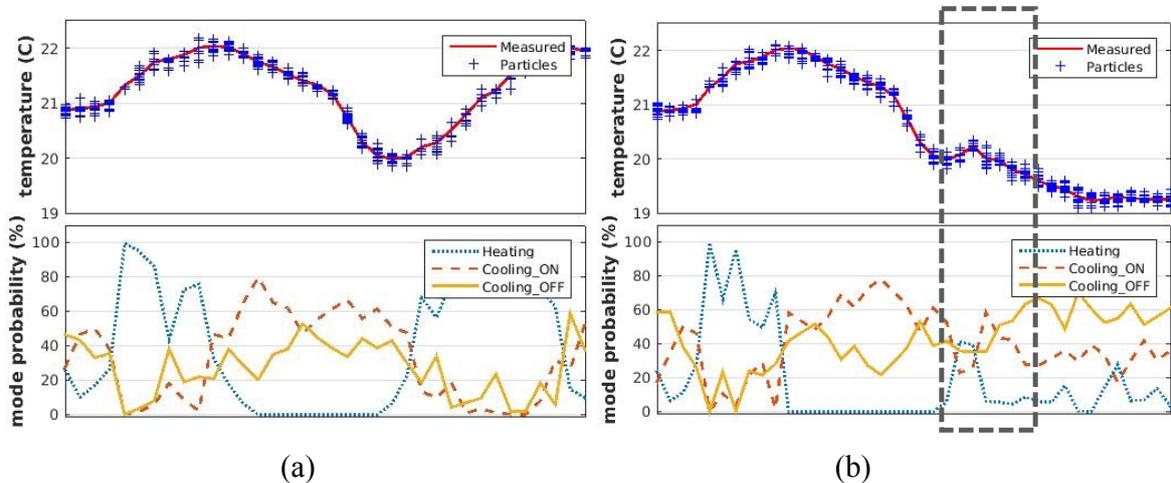


Abbildung 7.1.8: Empfindlichkeit bei Abweichungen der kontinuierlichen Größen

Zum Testen der Robustheit wird das System außerhalb des spezifizierten Temperaturregelungsintervalls (20 C° - 22 C°) geheizt bzw. gekühlt. Eine Überschreitung der oberen Grenze $T_{max} = 22\text{ C}^\circ$ wird in Abbildung 7.1.8 sowohl durch die Abweichung der

Partikelwerte von der Temperaturmessung als auch durch ein „Error“ im geschätzten Modus festgestellt.

7.2 Reduktion der Komplexität durch Unterräumen

Bei mobilen Robotern handelt es sich um die Klasse „Missionssysteme“. Für solche Systeme soll für jede Mission, basierend auf der Nutzereingabe, das entsprechende Überwachungsmodell zur Laufzeit automatisch generiert werden. Die Verwendung der Petri-Netze in ihrer grafischen Darstellung für die Roboternavigation erfordert eine a priori Erstellung der Modelle für alle möglichen Missionen. Abbildung 7.2.1 zeigt das Beispiel der Navigation eines mobilen Roboters in [Feh04]: Zur Modellierung des Navigationsprozesses wird die Umgebung in Zellen geteilt. Für jede Zelle werden Bedingungen für das Eintreten und Verlassen des Bereiches definiert.

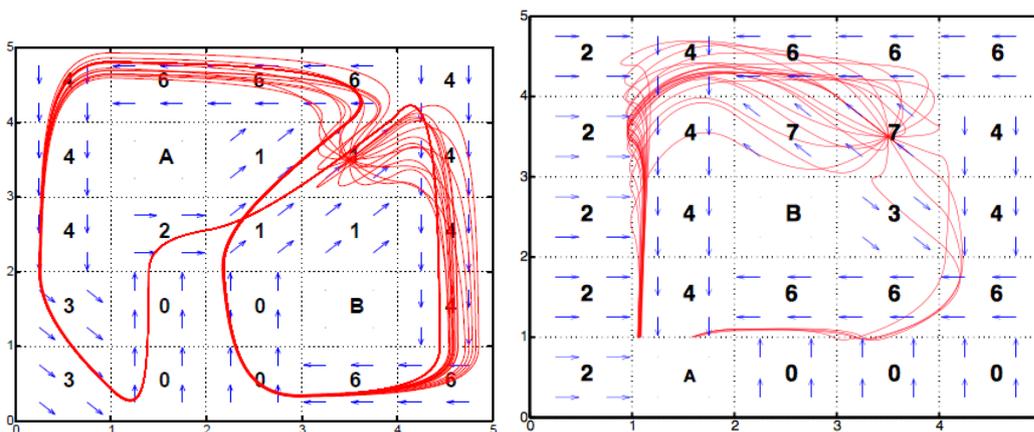


Abbildung 7.2.1: Gitterbasierte Navigation ([Feh04])

In einem Versuch durch [Les06], diesen Prozess mit Petri-Netzen zu modellieren, musste jede Zelle in der Umgebung durch eine Stelle im Petri-Netz dargestellt werden. Somit ergab sich allein für eine Fläche der Größe 4x4 ein Petri-Netzmodell mit 9 Stellen und 24 Transitionen, und für eine Navigationsinstanz der Größe 3x3 ergibt sich ein Petri-Netzmodell mit 16 Stellen und 48 Transitionen (Abbildung 7.2.2). Die Tabelle in Abbildung 7.2.3 zeigt die Dimensionen verschiedener Größen der Navigationsinstanzen und ein Beispiel für die Inzidenzmatrix der kleinsten Instanz.

Ein Modell zu verwenden, welches alle Zellen abdecken würde, wäre unübersichtlich und kostenineffizient bezüglich der Rechenzeit. Dieses Problem wurde in dieser Arbeit durch die

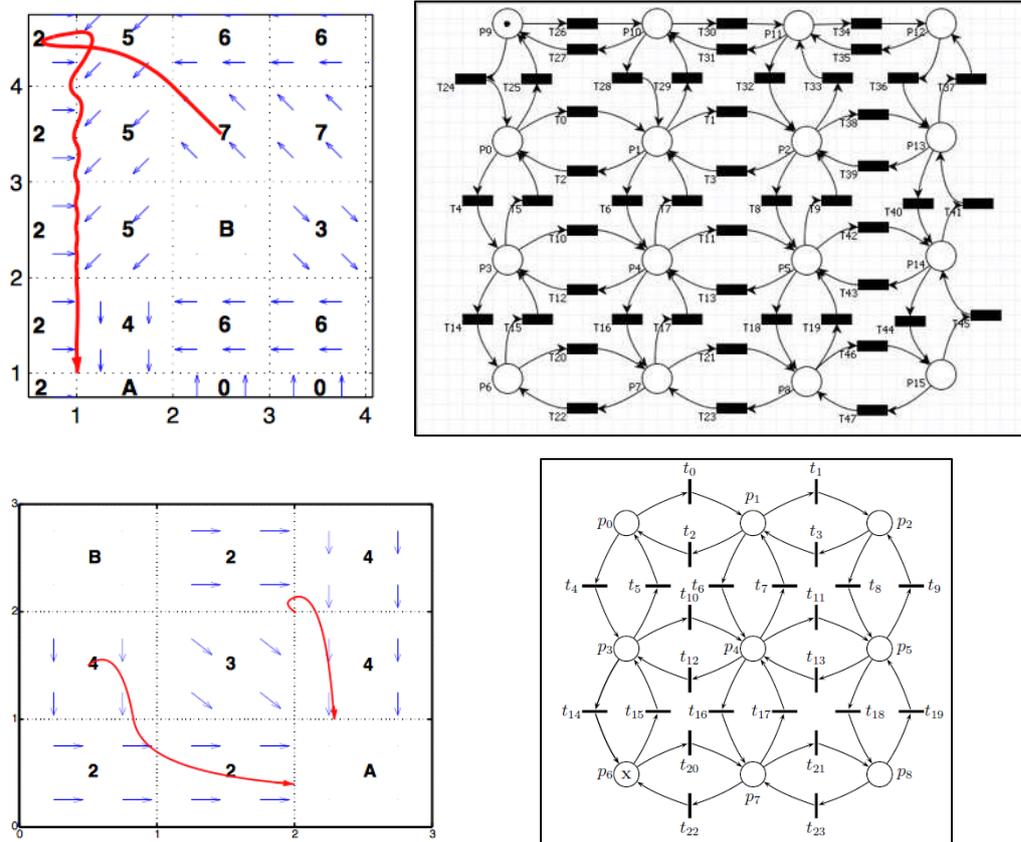


Abbildung 7.2.2: Modellierung der Gitterbasierten Navigation mittels Petrinetzen in ([Les06])

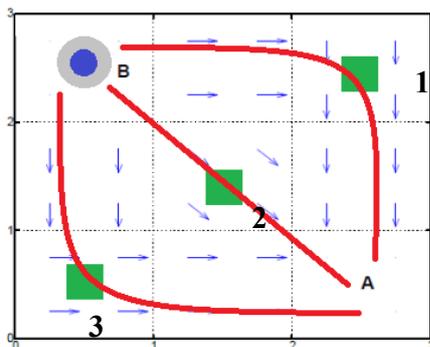
	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	G	$ P $	$ T $	$ D = P + T $
t_0	-x	x	0	0	0	0	0	0	0	3 × 3	9	24	33
t_1	0	-x	x	0	0	0	0	0	0				
t_2	x	-x	0	0	0	0	0	0	0				
t_3	0	x	-x	0	0	0	0	0	0	4 × 4	16	48	64
t_4	-x	0	0	x	0	0	0	0	0				
t_5	x	0	0	-x	0	0	0	0	0				
t_6	0	-x	0	0	x	0	0	0	0	5 × 5	25	80	105
t_7	0	x	0	0	-x	0	0	0	0				
t_8	0	0	-x	0	0	x	0	0	0				
t_9	0	0	x	0	0	-x	0	0	0	6 × 7	42	142	184
t_{10}	0	0	0	-x	x	0	0	0	0				
t_{11}	0	0	0	0	-x	x	0	0	0				
t_{12}	0	0	0	x	-x	0	0	0	0	9 × 9	81	288	369
t_{13}	0	0	0	0	x	-x	0	0	0				
t_{14}	0	0	0	-x	0	0	x	0	0				
t_{15}	0	0	0	x	0	0	-x	0	0	6 × 7	42	142	184
t_{16}	0	0	0	0	-x	0	0	x	0				
t_{17}	0	0	0	0	x	0	0	-x	0				
t_{18}	0	0	0	0	0	-x	0	0	x	9 × 9	81	288	369
t_{19}	0	0	0	0	0	x	0	0	-x				
t_{20}	0	0	0	0	0	0	-x	x	0				
t_{21}	0	0	0	0	0	0	0	-x	x	6 × 7	42	142	184
t_{22}	0	0	0	0	0	0	x	-x	0				
t_{23}	0	0	0	0	0	0	0	x	-x				

Abbildung 7.2.3: Inzidenzmatrix einer Navigationsinstanz der Größe 3x3

Beschreibung mittels BDF (Behavioural Description File) beseitigt, welches sich für die Online-Generierung der Missionsmodelle einsetzen lässt. Der Schwerpunkt dabei liegt in der Reduktion der Modelldimension, basierend auf der vorliegenden Situation bzw.

Nutzereingabe. Das resultierende Modell enthält somit nur noch die für die vorliegende Mission relevanten Informationen.

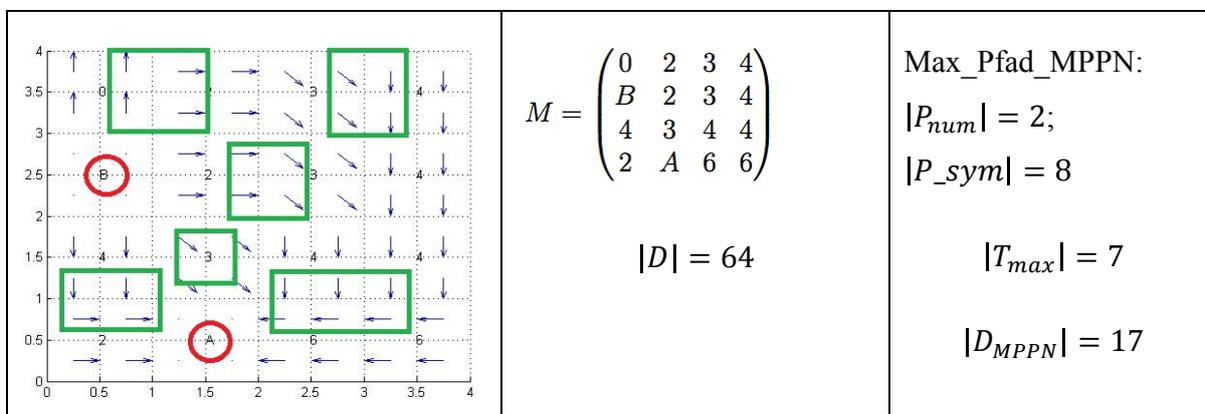
Der Vorteil der Zustandsraumreduktion ist die Vermeidung der kombinatorischen Explosion. Für die Analyse und Auswertung wird nicht der gesamte Zustandsraum aufgespannt und analysiert, sondern nur der für die vorliegende Situation relevante Teil. Abbildung 7.2.4 zeigt das Beispiel für die Navigationsinstanz (3x3). Ausgehend von einer Startposition B kann der Roboter sein Ziel A über einer dieser Trajektorien (rote Linien) erreichen. Die Auswahl des zu fahrenden Pfads hängt von der Initialausrichtung des Roboters ab. Der Modifizierte Petri-Netz (MPPN) für dieses Beispiel enthält die symbolischen Stellen $P^S = \{B, GOTO Area i, Area i, GOTO A\}$, die numerischen Stellen $P^N = \{Reached, Not Reached\}$ und die Transitionen $T = \{T_B, T_{Bi}, T_{iA}, T_A\}$.



MPPN (Nav.: 3x3)			
$ P_{num} $	$ P_{sym} $	$ T $	$ D $
2	4	3	9

Abbildung 7.2.4: MPPN basierte Modellierung einer Navigationsinstanz der Größe 3x3

Die Tabelle in Abbildung 7.2.5 zeigt weitere Beispiele von Navigationsinstanzen mit Vorschlägen zu Bereichen (in grün) für die Platzierung der Landmarken sowie die Ermittlung der Komplexität der MPPN-Modelle für den längsten Pfad zwischen dem Startpunkt B und dem Zielpunkt A.



	$M = \begin{pmatrix} 2 & 4 & 6 & 6 & 6 \\ 2 & 4 & 7 & 7 & 4 \\ 2 & 4 & B & 3 & 4 \\ 2 & 4 & 6 & 6 & 6 \\ 2 & A & 0 & 0 & 0 \end{pmatrix}$ $ D = 105$	Max_Pfad_MPPN: $ P_{num} = 2;$ $ P_{sym} = 7$ $ T_{max} = 6$ $ D_{MPPN} = 15$
	$M = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 2 & 4 & 4 & 6 & 6 & 6 & 6 \\ 2 & 4 & A & 6 & 6 & 6 & 6 \\ 2 & 4 & B & 1 & 7 & 7 & 0 \\ 2 & 4 & 2 & 1 & 7 & 7 & 0 \\ 2 & 2 & 2 & 1 & 1 & 0 & 0 \end{pmatrix}$ $ D = 184$	Max_Pfad_MPPN: $ P_{num} = 2;$ $ P_{sym} = 7$ $ T_{max} = 6$ $ D_{MPPN} = 15$
	$M = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & B & 2 \\ 0 & 1 & 2 & 3 & 4 & 5 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 \\ 2 & 2 & 2 & 2 & 2 & 2 & 3 & 4 & 5 \\ 0 & 0 & 3 & 3 & 3 & 2 & 3 & 4 & 6 \\ 0 & 0 & 3 & 3 & 3 & 3 & 3 & A & 6 \\ 1 & 1 & 2 & 2 & 3 & 3 & 3 & 0 & 0 \\ 1 & 1 & 2 & 2 & 2 & 2 & 2 & 0 & 0 \end{pmatrix}$ $ D = 369$	Max_Pfad_MPPN: $ P_{num} = 2;$ $ P_{sym} = 10$ $ T_{max} = 9$ $ D_{MPPN} = 21$

Abbildung 7.2.5: Beispiele von Navigationsinstanzen zur Berechnung der MPPN-Komplexität

7.3 Usability

Um die Übertragbarkeit des Verfahrens auf andere Anwendungen zu zeigen, wurde das Konzept in Zusammenhang mit weiteren Problemformulierungen und Anwendungen dargestellt.

a) Überwachung von Flugmissionen:

Das Konzept der Navigationsüberwachung mittels MPPN wurde auf das Beispiel eines fliegenden Roboters übertragen ([Mek12b]). Es handelt sich um einen unbemannten Hubschrauber am Lehrstuhl für Automation der Universität Heidelberg, welcher im

autonomen Flugmodus das Ziel durch das Anfliegen von Missionspunkten bzw. Wegpunkten erreicht (Abbildung 7.3.1). Diese Experimentierplattform „Aeroscout Scout B1-100“ verfügt über ein Sensorsystem, das aus einem Magnetometer, einer Trägheitsmesseinheit (IMU), einem Differential Global Positioning System (DGPS) und einer Kamera besteht.

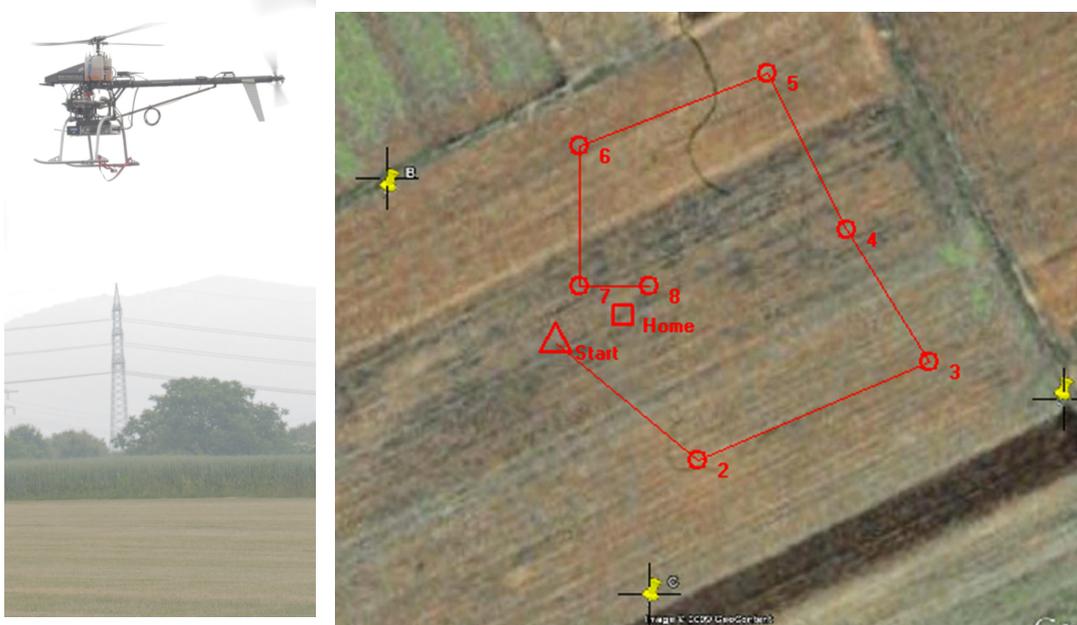


Abbildung 7.3.1: Planung einer Flugmission ([Sco09])

Die Dynamik lässt sich durch die Änderung des Zustandsvektors \mathbf{X} beschreiben, wobei:

$$\mathbf{X} = [u \ v \ w \ p \ q \ r \ \varphi \ \theta]^T$$

- $\mathbf{p} = [x \ y \ z]^T$: Positionsvektor des Schwerpunktes des Hubschraubers, ausgedrückt im universellen Koordinatensystem.
- $\mathbf{v} = [u \ v \ w]^T$: die Lineargeschwindigkeiten.
- $\boldsymbol{\omega} = [p \ q \ r]^T$: Roll-, Nick- und Gierwinkelgeschwindigkeiten.
- φ, θ : Euler Winkel.

Im Folgenden wird nur die Übertragung des Überwachungskonzeptes mittels MPPN eines mobilen Roboters auf einen Hubschrauber beschrieben, ohne dabei auf die Modellierung einzugehen. Details zum Modell sind in ([Mek12b] bzw. in der Literatur [Pad07] zu finden. Zur Anwendung der MPPN-Überwachung auf den Hubschrauber werden die Odometrie-Daten durch GPS, das RFID-System, durch die Kamera und die RFID-Tags durch künstliche Landmarken auf dem Flugfeld ersetzt. Der Nutzer definiert einen Zielpunkt „Goal“. Basierend darauf, wird der Missionsplan automatisch generiert. Abbildung 7.3.2 zeigt sowohl eine Aufnahme als auch eine schematische Darstellung einer Flugmission, bei der die

Wegpunkte WP1→WP2→WP3→WP4 angefliegen werden müssen, um das Ziel „Goal“ zu erreichen.

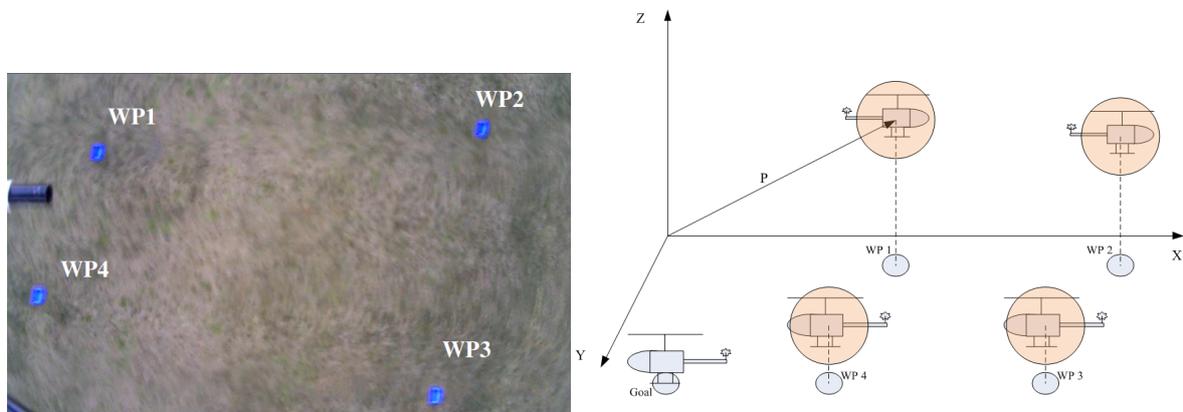


Abbildung 7.3.2: Missionspunkte für die Überwachung ([Mek12b])

Das MPPN-Modell dieses fliegenden Roboters unterscheidet sich vom mobilen Roboter nur in den Bewegungsgleichungen, die vom numerischen Teil des Modells verwendet werden und den Bedingungen für das Eintreten und Verlassen einer Area. Beim Hubschrauber sind die Areas als Kugeln modelliert, am Beispiel des Rollstuhls als Flächen (x, y) . Die Form der Area (Abbildung 7.3.3) ist eine Modellierungsannahme, die von der Anwendung abhängt.

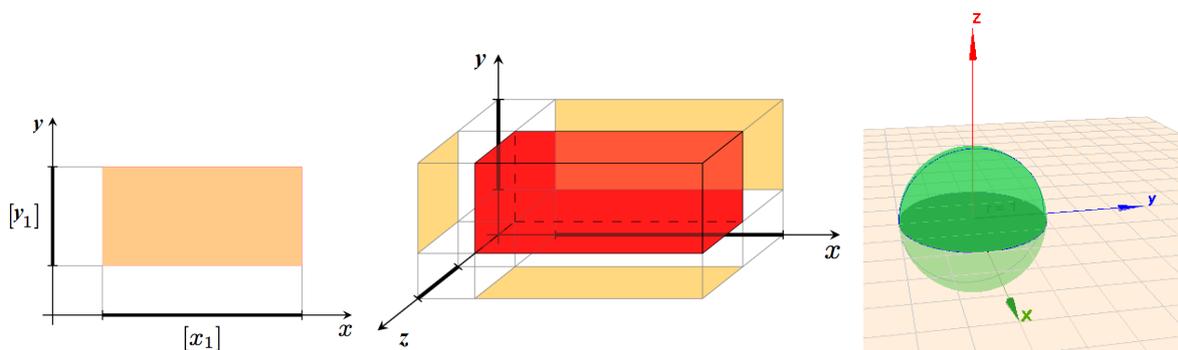


Abbildung 7.3.3: Modellierung der Detektionsbereiche

Das MPPN-Modell für die Flugmission (Abbildung 7.3.4) enthält folgende Elemente:

- $P_S = \{WP1, GOTO WP2, WP2, GOTO WP3, WP3, GOTO WP4, WP4, GOTO Goal\}$.
- $P_N = \{Reached, Not Reached\}$
- $T = \{T_1, \dots, T_7 \mid T_i \in T^H, i = 1..7\}$
- Zustandsvektor: $X = \{x, y, z \mid x, y, z \in \mathfrak{R}\}$
- Die Differentialgleichungen für die numerischen Stellen beschreiben die zwei Zustände: $\{Reached, Not Reached\}$.

Die Konfigurationen in den symbolischen Stellen: $\gamma(WP_N) = N$, $\gamma(GOTO WP_N) = 0$ mit $N = 1..L$, und L ist die Anzahl der Wegpunkte.

- Die Bedingungen an den numerischen Transitionen beschreiben ein kugelförmiges Gebiet:

$$\Omega^N(T_i) = \{x_i - \varepsilon \leq x \leq x_i + r \wedge y_i - \varepsilon \leq y \leq y_i + r \wedge z_i - \varepsilon \leq z \leq z_i + r \}$$

- Bedingungen an den symbolischen Stellen:

$$\Omega^S(T_1) = \Omega^S(T_3) = \Omega^S(T_5) = 0, \Omega^S(T_2) = 2, \Omega^S(T_4) = 3 \text{ und } \Omega^S(T_6) = 4$$

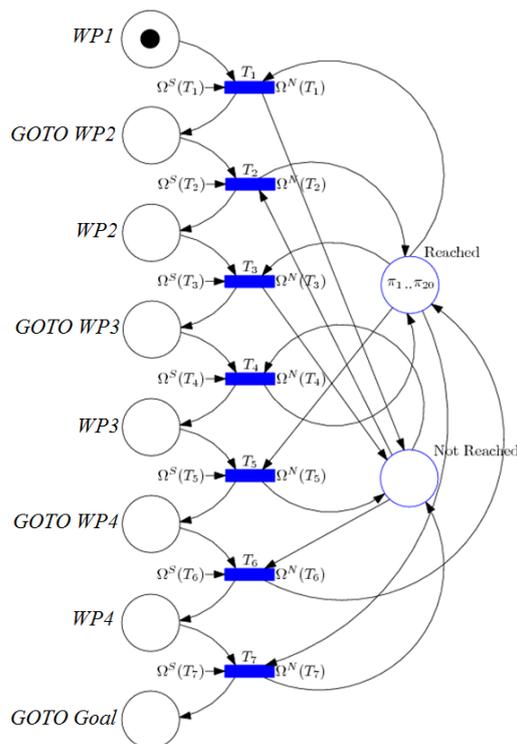
- Anfangsmarkierung:

$$M_0 = (20\pi \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0), \text{ mit } M_0 = \Pi_0 \cup \Delta_0, \Pi_0 = \{\pi_i^0 \mid i = 1..100\} \text{ und}$$

$$\pi_i^0 = \left[\{x_0, y_0, z_0\}, \frac{1}{100} \right]$$

- Inzidenzmatrix:

$$C = \begin{bmatrix} -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$



```
>Netparameters 3
[x y z]
>Configurations 1
[area]
>Max_Deviations 3
[eps_x] [EPS=E_x]
[eps_y] [EPS=E_y]
[eps_z] [EPS=E_z]
>Places 10
[Reached] [NUM] [A] [B]
[Not Reached] [NUM] [A] [B]
[WP1] [SYM] [1.0]
[GOTO WP 2] [SYM] [0.0]
[WP 2] [SYM] [0.0]
[GOTO WP 3] [SYM] [0.0]
[WP 3] [SYM] [1.0]
.....
>Transitions 7
[T1] [HYB] [C] [Q_N(T1), Q_S(T1)]
[T2] [HYB] [C] [Q_N(T2), Q_S(T2)]
[T3] [HYB] [C] [Q_N(T3), Q_S(T3)]
[T4] [HYB] [C] [Q_N(T4), Q_S(T4)]
.....
>Incidence matrix
-1 1 0 -1 0
1 -1 -1 0 1
0 0 -1 -1 1
0 0 1 1 -1
0 0 1 1 -1
.....
>Pred_noise [v]
>Obs_noise [u]
>Marking101
[Reached] [NUM] [x0,y0,z0]
[Reached] [NUM] [x0,y0,z0]
[Reached] [NUM] [x0,y0,z0]
.....
[Area 1] [SYM] [1.0]
```

Abbildung 7.3.4: MPPN-Modell für die Flugüberwachung ([Mek12b])

b) Integration von MPPN im Navigationsgedächtnis mobiler Roboter:

Eine weitere Anwendung der MPPN wurde in der Arbeit [Ale12] gezeigt. Dabei wurde das hybride Überwachungskonzept der MPPN durch einen assoziativen Speicher erweitert (Abbildung 7.3.5). Dieser Speicher ermöglicht es dem Roboter, sich an einen zuvor erfolgreich gefahrenen Weg zu erinnern bzw. neue unbekannte Wege online zu lernen. Dieses Konzept macht die Überwachung zuverlässiger und robuster in Bezug auf die Fehlererkennung, -diagnose und -behandlung, da beim Auftreten von Fehlern der Navigationsprozess nicht sofort abgebrochen wird. Dadurch wird die Anzahl der vom Diagnosemodul ausgelösten Alarme reduziert.

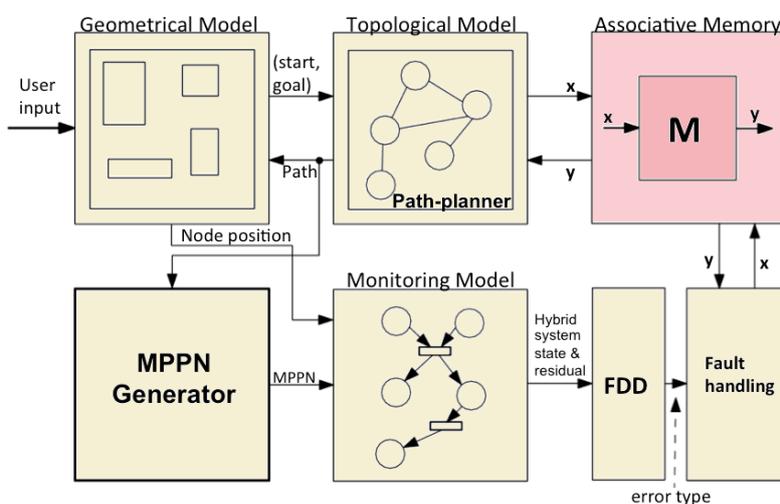


Abbildung 7.3.5: Integration des Assoziativspeichers im Überwachungskonzept ([Ale12])

Der Kern der Assoziativspeicher ist die Matrix M . Das Eingangsmuster dieser Speicher ist ein Vektor x . Aus der Multiplikation von x mit M ergibt sich der Ausgangsvektor $y = M \cdot x$. Die Realisierung dieser Assoziativspeicher ist in Abbildung 7.3.6 dargestellt. Der Eingangsvektor ergibt sich aus dem Produkt des Referenzvektors r und der Verbindungsmatrix C .

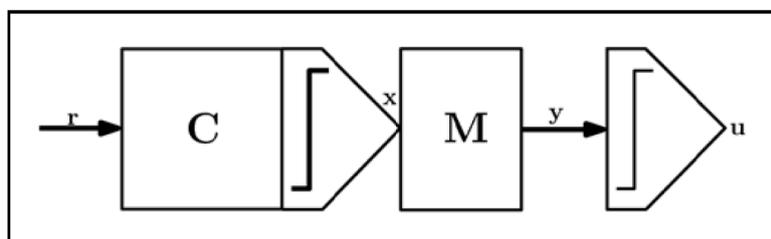


Abbildung 7.3.6: Realisierung der Assoziativspeicher ([Ale12])

Am Beispiel der Navigation werden der Start- und der Zielpunkt ausgewählt und daraus wird der Referenzvektor r ermittelt. Mit der Multiplikation des berechneten Pfades r durch die Verbindungsmatrix C , die Beziehungen zwischen den Navigationsknoten in der Umgebung

beschreibt, wird das Eingangsmuster x gewonnen. Durch $M \cdot x$ wird der Speicher abgerufen („recall“): Entspricht das Ergebnis dieser Berechnung einem bereits bekannten Pfad, wird der Pfadplanungsalgorithmus ausgeführt. Ist das Ergebnis unbekannt, wird der Pfad durch den Graph-Such-Algorithmus durchsucht und dem Assoziativspeicher gelehrt („learning“).

Der Nutzen der Integration des assoziativen Speichers im Überwachungskonzept lässt sich am Beispiel der Navigation durch das folgende Szenario zeigen: Angenommen, dass der Roboter, ausgehend von der Ausgangsknote 1, bereits drei Wege erfolgreich gefahren ist, dann werden die folgenden Pfade dem Speicher gelehrt:

- $1 \rightarrow 3 \rightarrow 5: r = [1, 0, 1, 0, 1, 0, 0]$
- $1 \rightarrow 3 \rightarrow 4 \rightarrow 7: r = [1, 0, 1, 1, 0, 0, 1]$
- $1 \rightarrow 2 \rightarrow 4 \rightarrow 7: r = [1, 1, 0, 1, 0, 0, 1]$

Nun ist die vorliegende Mission von Area 1 zu Area 5 zu navigieren. Während der Navigation erwartet der Überwacher als nächstes die Detektion der Area 3. Dies ist allerdings nicht geschehen. In der klassischen Überwachung wird die Navigation an dieser Stelle abgebrochen. Durch die Erweiterung um den Assoziativspeicher startet der Überwacher einen „Recall“, um zu überprüfen, ob der Pfad bzw. dieser Teilpfad bereits durch den Roboter fehlerfrei gefahren wurde. Die Antwort auf diesen „Recall“ ist: $1 \rightarrow 3 \rightarrow 5$. Der Roboter fährt weiter, und der Überwacher erwartet eine Detektion der Area 5. Wird diese detektiert, wird die Fehlermeldung mit dem Area 3 zurückgesetzt. Ansonsten wird der Navigationsprozess abgebrochen. Abbildung 7.3.7 ([Ale12]) zeigt das gesamte Konzept, am Beispiel der Navigationsüberwachung.

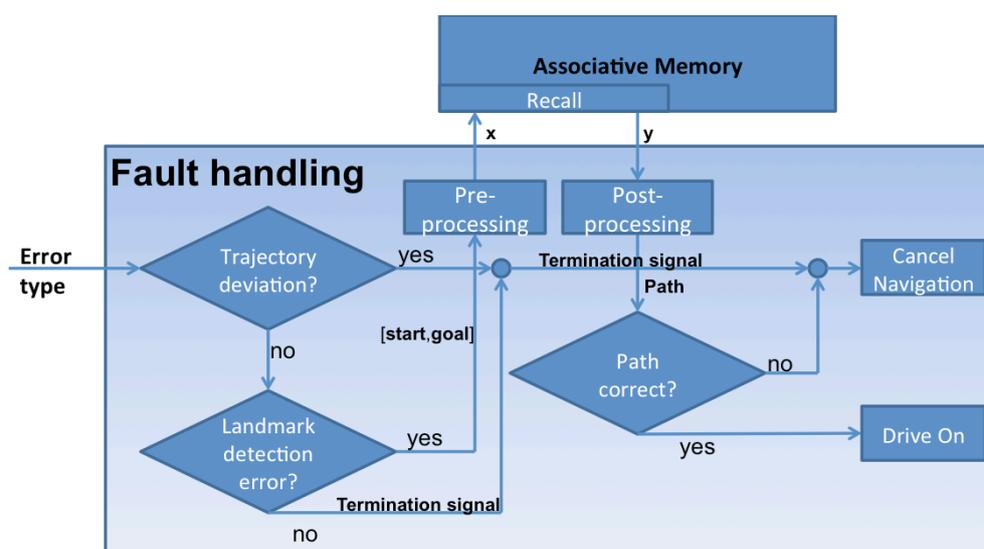


Abbildung 7.3.7 Rolle des Assoziativspeichers im Fehlerdiagnosemodul

8 Fazit und Ausblick

Ein Fehler in einer Komponente kann zur Reduktion der Gesamtleistung des Systems und zu einer inakzeptablen Verschlechterung der Systemfunktionalität bzw. Systemverlässlichkeit führen. Die Funktionalität und die Performance des gesamten Systems hängen somit von der Performance der einzelnen Komponenten ab. Daher ist es wichtig, Systeme durch Überwachungskomponenten, die das Auftreten von Fehlern automatisch und früh erkennen, zu erweitern. Diese Überwachungskomponenten sollten sowohl in der Entwicklungsphase als auch während des Betriebs eingesetzt werden. Um herauszufinden, ob sich ein System korrekt und wie beabsichtigt verhält, werden während der Entwicklung verschiedene Tests und Verifikationprozesse, die die Spezifikation und die Realisierung überprüfen, durchgeführt. Ist ein korrektes System realisiert und zeigen die Tests, dass die Spezifikation erfüllt ist, ist dies jedoch keine Garantie, dass sich das System während des Einsatzes in einer realen Umgebung weiterhin richtig verhält. Denn die Interaktion eines technischen Systems mit einer realen Umgebung ist nur partiell kontrollierbar und modellierbar, Ereignisse sind nicht verlässlich voraussagbar, und während des Betriebs ist das System mit einer Umgebung konfrontiert, worüber es nur zum Teil eine interne Repräsentation besitzt. Diese Wechselwirkungen mit der Umgebung bzw. mit weiteren technischen Systemen könnte das Verhalten des Systems beeinflussen und zu einer „Degradation“, d. h. im Gegensatz zu einem Systemausfall, einer verminderten Funktionstüchtigkeit führen. Somit ist eine On-line Überwachung und eine laufende Überprüfung des Systemverhaltens gegenüber seiner Spezifikation während des Betriebs erforderlich.

Aus diesem Grund war der Fokus in dieser Arbeit, die Verhaltensspezifikation der Komponenten in die Überwachung einzubetten, um trotz der Ungewissheiten und der äußeren Störeinflüsse herauszufinden, ob das System seine Spezifikation erfüllt und sich wie erwartet verhält. Hierfür wurde ein einheitliches Verfahren für die Verhaltensmodellierung und -überwachung entwickelt, welches eine nahtlose Überwachung während der Entwicklung und während des Betriebs ermöglicht. Grundgedanke dieses Konzeptes war die Realisierung der Überwachungskomponente durch die Wiederverwendung der ursprünglichen Verhaltensspezifikation der zu überwachenden Komponente, damit diese während der Entwicklung und in der Betriebsphase mit ihrer Spezifikation verglichen werden kann. Aus diesem Grund wurde die Überwachungskomponente anhand Kobra einheitlich, wie alle anderen Systemkomponenten, spezifiziert, realisiert und in einer frühen Phase der Entwicklung in Teststrukturen eingebettet. Für die Verhaltensbeschreibung wurden die Petri-

Netze als formale Methoden eingesetzt, um eine Verifikation der entwickelten Modelle in einer frühen Phase der Systementwicklung durchführen.

Der Fokus im ersten Teil der Arbeit lag auf der Übertragung der verhaltensbasierten Modellierung von der Spezifikation in die Überwachung und auf der systematischen Einbettung der Überwachung in allen Schritten der komponentenbasierten Entwicklung nach KobrA. Die generische Designmethode wurde auf eine generische Systemarchitektur, die mit dem Top-down Design und der Bottom-up Implementierung kompatibel ist, angewendet. Somit wurde das System in Teilsysteme zerlegt, und für jedes Teilsystem wurde eine Netzspezifikation (Spezifikation mit Hilfe eines Petri-Netzes) erstellt, welche einen Teilentwurf des Gesamtsystems darstellte und zum Schluss zu einer Überwachungskomponente resultierte.

Da es sich in dieser Arbeit um hybride Systeme handelt, wurde das hybride Verhalten der zu überwachenden Komponenten durch Petri-Netze als diskrete Modelle und Differentialgleichungen bzw. Differenzengleichungen als kontinuierliche Modelle repräsentiert. Diese Verkopplung von ereignisdiskreten und kontinuierlichen Modellen wurde hier durch eine neue Klasse von hybriden Petri-Netzen, so genannte Modifizierte Partikel-Petri-Netze (MPPN), modelliert. Die MPPN modellieren dynamische Systeme unter Berücksichtigung folgender Aspekte: Die Struktur des Ansatzes, die einheitliche Modellierung auf den verschiedenen Abstraktionsebenen und die Berücksichtigung der dynamischen Wechselwirkungen zwischen beiden Teilen der Dynamik, kontinuierlich und diskret.

Zur Erzielung einer robusten Überwachung gegenüber Ungewissheiten, bedingt durch die Vereinfachungen der verwendeten Systemmodelle sowie der Störungen bei der Messdatenerfassung, wurde der probabilistische Framework vorausgesetzt. Im numerischen Teil wurde eine Menge von Partikeln gezogen, die sich nach Differenzialgleichungssystemen entwickelt haben. Sie bilden den Prädiktionsschritt für die Zustandsschätzung. Im diskreten Teil wurden spezielle Regeln für die Feuerbarkeit der Transitionen entwickelt, welche auf dem Konzept der Makromarkierung (alle Transitionen, die schalttfähig sind, feuern gleichzeitig) und der Pseudo-Feuerbarkeit (Kopieren der Marken anstatt zu löschen, um die Ungewissheit der Beobachtung zu berücksichtigen) basieren, um damit eine aktuelle Menge möglicher Folgezustände zu präzisieren. Im anschließenden Korrekturschritt wurde anhand einer symbolischen und einer numerischen Beobachtung, die ebenfalls unsicherheitsbehaftet

ist, der höchstwahrscheinlichste hybride Systemzustand ermittelt. Die Schätzung dieser Zustände, die Modellierung der Wechselwirkungen beider Teile des Modells sowie der stochastische Rahmen dieser Analyse bilden den theoretischen Schwerpunkt dieser Arbeit.

Für eine zuverlässige und frühzeitige Erkennung von Fehlern, bei der weder vorhandene Fehler übersehen noch falsche Alarme ausgelöst werden, wurde in dieser Arbeit eine Konsistenzanalyse durchgeführt. Diese Analyse basiert auf einer Gegenüberstellung des Residuums vom wert-kontinuierlichen Systemzustand zu den möglichen Markierungen des Petri-Netzes und auf einer Untersuchung der Erreichbarkeit der diskreten Zustände. Dadurch wurden Abweichungen und Inkonsistenzen zwischen dem modellierten und dem tatsächlichen Verhalten des betrachteten Systems festgestellt und auf die entsprechenden Fehlerklassen zurückgeführt.

Für die verschiedenen Schritte des Entwicklungsprozesses wurden Werkzeuge benötigt. Für die Erstellung von formalen Modellen für die Spezifikation wurde ein Petri-Netzeditor weiterentwickelt. Um die Realisierung der Überwachungskomponente direkt aus der Spezifikation generieren zu können, wurde der Petri-Netzeditor durch einen Kompilierer erweitert. Aus dem Petri-Netzmodell ergibt sich eine XML-basierte Verhaltensbeschreibung, der so genannte „Behavioural Description File“ (BDF). Dieses Verhaltensmodell bildet die Grundlage zur Ableitung des in dieser Arbeit entwickelten Überwachungsmodells, welches vom implementierten Simulator ausgeführt wird.

Das Verfahren wurde sowohl auf prozesstechnische Systeme (Heater Control System, HCS) sowie auf autonome mobile Roboter (autonomer Rollstuhl) angewendet. Anhand dieser unterschiedlichen Anwendungsbeispiele wurde das MPPN-basierte Überwachungs- und Diagnoseverfahren veranschaulicht und evaluiert. Die Ergebnisse der Evaluation zeigen die Vorteile des entwickelten Verfahrens, in Bezug auf die Robustheit, Empfindlichkeit, Komplexitätsreduktion und Anwendbarkeit in verwandten Forschungsthemen. Die Ergebnisse zeigen, dass das Verfahren unempfindlich gegenüber Rauschen und Modellunsicherheiten ist, aber in Bezug auf Fehler empfindlich reagiert. Diese Empfindlichkeit des Diagnosemoduls wurde durch die Unsicherheitsintervalle (ϵ_{\max} und ϵ_{Safe}) aus der Spezifikation beeinflusst, die daher einerseits so klein wie möglich gewählt wurden, andererseits aber in jedem Fall die tatsächlichen Unsicherheiten abgedeckt haben. Bei komplexen Analyse- und Entwurfsproblemen stellt sich die Frage, wie die Komplexität zu reduzieren ist? Dieses Problem wurde an verschiedenen Stellen der Arbeit behandelt. Zum

einen erfolgte eine Zustandsraumreduktion durch das Konzept der hierarchischen Modellierung und Zustandsschätzung, was die kombinatorische Explosion des Zustandsvektors verhindert. Zum anderen wurde die Komplexität durch die Verwendung von Unterräumen reduziert. Denn durch die On-line Generierung der Modellstruktur bzw. Modellparameter für die vorliegende Mission, wurde für die Analyse und Auswertung nicht der gesamte Zustandsraum aufgespannt und analysiert, sondern nur der für die vorliegende Situation relevante Teil.

Die praktische und theoretische Anwendbarkeit des entwickelten Verfahrens lässt sich durch dessen Einsetzbarkeit in anderen Forschungsarbeiten verdeutlichen. In [Sad14] macht die Autorin Gebrauch von den Vorteilen der hier vorgeschlagenen hierarchischen Zustandsschätzung und hat den Partikel-Filter durch Kalman-Filter ersetzt. Dabei ging es um eine kaskadierte Struktur von Kalman-Filtern für die Fehlerschätzung in einem optischen Navigationssystem. Der Einsatz von MPPN in anderen Anwendungsgebieten, wie z. B. in Zusammenhang mit Testsheets in [Atk10] oder für das Verlässlichkeit-Assessment in [Wag12], zeigen, dass sich dieses Verfahren nicht nur auf die Fehlerdiagnose als Hauptziel der Überwachung beschränkt, sondern auch für die Analyse einiger Sicherheitseigenschaften des Systems (Verlässlichkeit, Performance, Safety) nützlich ist. In [Gau14] und [Gau15] wurden die MPPN für die Überwachung eines Wassertanksystems eingesetzt. Das Verfahren wurde in drei verschiedenen Szenarien getestet. Die Ergebnisse waren konsistent mit den Erwartungen des Autors, und die Vorteile des Verfahrens waren motivierend für eine zukünftige Implementierung in einem realen Wassertanksystem.

Ein interessanter Vorschlag für eine zukünftige Erweiterung der hier entwickelten Überwachungs- und Fehlerdiagnoseverfahren ist die Verknüpfung mit Rekonfigurationsmechanismen. Hierfür soll das Petri-Netzmodell für die Überwachung einer Komponente mit einem weiteren Petri-Netz verbunden werden, welches eine Abfolge von Zuständen enthält. Die Ausführung dieser Zustände soll Korrekturmaßnahmen einleiten, die das System von einem sicherheitskritischen Zustand zu einem sicheren Zustand überführen. Der Wechsel vom Überwachungsmodell zu einem Rekonfigurationsmodell wird durch das Diagnoseergebnis getriggert und durch das Feuern einer Rekonfigurationstransition t_R , die beide Petri-Netzmodelle verknüpft, implementiert. Die Anwendung des Rekonfigurationskonzeptes auf die Roboternavigation entspricht der Korrektur der Roboterposition, wenn beim Überfahren eines RFID-Tags der Vergleich zwischen der Soll-Position auf dem RFID-Tag und der durch die Odometrie gemessene Position eine

Abweichung aufweist. Anstatt wegen dieser Abweichung den Navigationsprozess abubrechen, korrigiert der Roboter seine Abweichung durch einige Fahrmanöver, die diesen Positions-Offset auf null zurücksetzen. Durch diese Korrekturmaßnahmen wird vermieden, dass der Positionsfehler sich während der Fahrt akkumuliert. Am Beispiel eines Temperaturregelungssystems sollte eine Überhitzung dazu führen, dass der Überwacher einen Zustandswechsel im Petri-Netzmodell auf den Modus „Overheating“ erkennt. Dieser Zustand soll der Trigger für eine Rekonfiguration sein, welche Kühlmaßnahmen auslöst; so kann das Controller-Board den Motortreiber bei Überhitzung ausschalten und die Lüfter aktivieren. Dieses Konzept lässt sich auf viele andere typische Systeme in der Literatur übertragen (z. B. Überwachung des Füllstands und Ventilöffnung in einem Wassertanksystem, Überwachung von Fahrassistenzsystemen und die Selektion des für die vorliegende Fahrsituation am geeignetsten Fahrmodus).

Mit den Methoden und Konzepten, die in dieser Arbeit entwickelt wurden, stehen für Forschungsgebiete im Ingenieurwesen neue theoretische sowie praktische leistungsfähige Werkzeuge bereit, die bei der Verhaltensmodellierung, Überwachung und Diagnose von hybriden dynamischen Systemen eingesetzt werden können und sich auf weitere praktische Anwendungen leicht übertragen lassen.

Insofern tragen die Ergebnisse dieser Arbeit zur Optimierung technischer Systeme bei. Als Grundlage für zukünftige Forschungen im Ingenieurwesen können die entwickelten Verhaltens-, Prozess- und Diagnosemodelle als Basis für die Fortentwicklung technischer Systeme dienen.

Liste der Abkürzungen

AMR	Autonome mobile Roboter
BDF	Behavior Description File
CBD	Komponentenbasierte Ententwicklung (engl., Component Based Development)
CUM	Component under Monitoring
DGPS	Differential Global Positioning System
EXT-PIPE	EXTended Platform Independent Petri net Editor
FA	Finite automata
FSM	Finite state machines
GUI	Graphical User Interface
HA	Hybride Automaten
HCS	Heater Control System
HPN	Hybride Petri-Netzen
HTCS	House Temperature Control System - Temperaturregelungssystem des Hauses.
IMU	Inertial measurement unit - Inertiale Messeinheit
MFD	Monitoring and Fault Detection
MPPN	Modifizierte Partikel-Petri-Netze“ (engl., „Modified Particle Petri Nets“)
OCL	Object Constraint Language
PC	Position Control
PI	PI-Reglers (proportional–integral controller)
PIPE	Platform Independent Petri net Editor
PP	Path Planning
PPN	Partikel-Petri-Netz PPN
PNML	Petri Net Markup Language
PWM	Pulsweitenmodulationen (engl., Pulse Width Modulation)
RCS	Room-Temperature Control System - Raumtemperaturregelungssystem
RFID	Radio Frequency Identification
RL	Robot Level
RNBC	R ecursive N ested B ehaviour-based C ontrol
SMC	Sequentiellen Monte Carlo
SIS	Sequential Importance Sampling
SIR	Sampling Importance Resampling

UML	Unified Modeling Language
WP	Waypoint
XML	EXtensible Markup Language

Literaturverzeichnis

- [Abd06] Abdel-Geliel Shahin, Mostafa A. M.: Fault Diagnose and Performance Recovery Based on the Dynamic Safety Margin, Inauguraldissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften der Universität Mannheim, Universität Mannheim, Mannheim, 2006.
- [Ake04] Akerholm M. and Fredriksson J., “A Sample of Component Technologies for Embedded Systems,” Technical Report, Mälardalen Research and Technology Centre, Mälardalen University, Västerås, Sweden, 2004.
- [Ale01] A. Alessandri, P. Coletta, Design of Luenberger observers for a class of hybrid linear systems, in: Hybrid Systems: Computation and Control, in: Lecture Notes in Computer Science, vol. 2034, Springer-verlag, 2001, pp. 7_18.
- [Ale11] A. Alexopoulos: “Überwachung des Navigationsprozesses eines autonomen mobilen Roboters basierend auf Petrinetzen und Monte-Carlo Verfahren”. Diplomarbeit am Lehrstuhl für Automation (Universität Heidelberg) unter Betreuung von Dipl. Ing. Leila Mekacher und Prof. Badreddin (2011).
- [Ale12] Alexopoulos, A., L. Mekacher Zouaghi and E. Badreddin, Associative memory for modified Petrinet based monitoring of mobile robot navigation. 1st International Conference on Innovative Engineering Systems (ICIES), Alexandria, Egypt, December, 7-9, 2012.
- [Ale14] Alexander Alexopoulos, Tobias Schmidt and Essameddin Badreddin. Pursuit and evasion in a recursive nested behavioral control structure for unmanned aerial vehicles. In 14th International Conference on Control, Automation and Systems (ICCAS). pp 1175-1180, Oktober 2014.
- [Alk12] A. Alkazaz: „Entwurf und Implementierung eines Pfadplanungsverhaltens für einen autonomen Rollstuhl“. Bachelorarbeit am Lehrstuhl für Automation (Universität Heidelberg) unter Betreuung von Dipl. Ing. Leila Mekacher, Prof. Badreddin und Prof. Suhr (Hochschule Mannheim) (2012).
- [All98] Alla, H.; David, R.: Continuous and Hybrid Petri Nets. Journal of Circuits, Systems & Computers 8 (1998) 1, S. 159-188.
- [Ame05] A. Ames, A. Abate et S. Sastry : Sufficient conditions for the existence of Zeno behavior. In IEEE Conference on Decision and Control, Séville, Espagne, 2005.
- [Ant00] P.J. Antsaklis, Hybrid Systems: Theory and Applications, a Brief Introduction to the Theory and Applications of Hybrid Systems, Proceeding of the IEEE 88 (7) (2000) 879_889 (special issue).

- [App07] Appel, A.; Herold, S.; Klus, H.; Rausch, A.: Modelling the CoCoME with DisCComp. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 267 – 296
- [Ari11] H Aris, S Salim (2011) State of Component Models Usage : Justifying the Need for a Component Model Selection Framework International Arab Journal of Information Technology 8: 3. 310-317
- [Aru02] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/non-gaussian Bayesian tracking,” IEEE Transactions on Signal Processing, vol. 50, no. 2, pp. 174–188, 2002.
- [Atk00] C. Atkinson, J. Bayer, D. Muthig (2000). Component-Based Product Line Development: The Kobra Approach. Proceedings of the First Software Product Line Conference, pp. 289-309.
- [Atk02a] Atkinson, Colin ; Bayer, Joachim ; Bunse, Christian ; Kamsties, Erik ; Laitenberger, Oliver ; Laqua, Roland; Dirk Muthig, Barbara Paech, Jürgen Wüst, Jörg Zettel. “Component-based Product Line Engineering with UML”, ISBN 0-201-73791-4, 2002.
- [Atk02b] Atkinson, C., and H.G. Gross. 2002. „Built-in contract testing in model-driven component-based development“. The Seventh International Conference on Software Reuse (ICSR-7), Austin, TX, April 15-19, 2002.
- [Atk03a] C. Atkinson, H. Bär, J. Bayer, C. Bunse, J.-F. Girard, H.-G. Gross, S. Kettemann, R. Kolb, T. Kühne, T. Romberg, O. Seng, P. Sody, E. Tolzmann. Handbuch zur komponentenbasierten Software Entwicklung. Fraunhofer/FZI Forschungs_zentrum, 2003.
- [Atk03b] C. Atkinson, H.-G. Groß, F. Barbier, Chapter 8, "Component Integration through Built-in Contract Testing in Component-Based Software Quality: Methods and Techniques", Lecture Notes in Computer Science, Springer, 2003, pp. 159-183.
- [Atk05] C. Atkinson, C. Bunse, H.-G. Gross, C. Peper (Eds). Component-Based Software Development for Embedded Systems. Lecture Notes in Computer Science, LNCS 3778, Springer, Heidelberg, 2005
- [Atk07] Atkinson, C. ; Bostan, P.; Brenner, D.; Falcone, G.; Gutheil, M.; Hummel, O.; Juhasz, M.; Stoll, D.: Modeling Components and Component-Based Systems in Kobra. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 54 – 84
- [Atk08] Atkinson, C.: Component-Oriented Verification of Software Architectures through Built-in Tests. Second European Conference, ECSA 2008 Paphos, Cyprus, September 29-October 1, 2008 Proceedings
- [Atk10] Atkinson C., Barth F. and Falcone G. (2010). Software testing using test sheets. International Workshop on Test-driven Development. Co-located with ICST 2010 Paris, France.

- [Atk14] C. Atkinson, R. Gerbig, K. Markert, M. Zrianina, A. Egunov, and F. Kajzar, "Towards a deep, domain specific modeling framework for robot applications," in Proceedings of the First Workshop on Model-Driven Robot Software Engineering (MORSE). CEUR-WS, 2014. [Online]. Available: <http://ceur-ws.org/Vol-1319/>
- [Bad90] E. Badreddin. Associative memory implementation in path-planning for mobile robots. In Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on, pages 14 –19 vol.1, May 1990.
- [Bad91] Badreddin, E.: "Recursive Behaviour-based Architecture for Mobile Robots". Robotics and Autonomous Systems, vol. 8, 1991.
- [Bad94] Badreddin, E.: "Control Structure and Behaviour-fusion for Autonomous Mobile Robots". International Journal of Robotics and Automation, vol.9, Issue 2, 1994.
- [Bad97] E. Badreddin. Control and system design of wheeled mobile robots. Habilitation script, 1997.
- [Bal02] A. Balluchi, L. Benvenuti, M.D. Di Benedetto, A.L. Sangiovanni-Vincentelli, Design of observers for hybrid system, in: Hybrid Systems: Computation and Control, in: LNCS, vol. 2289, Springer-Verlag, 2002, pp. 76_89.
- [Bal11] Ballarini P. et al., "Petri Nets Compositional Modeling and Verification of Flexible Manufacturing Systems", In 7th Annual IEEE Conference on Automation Science and Engineering (CASE 2011), 2011.
- [Bar92] Barroca, L., McDermond, J.: Formal methods: use and relevance for the development of safety critical systems. Computer journal, 35(6), 579-599.
- [Bar03] Bartolein, Christian: Entwurf und Aufbau eines Steuerungssystems zur Kollisionsvermeidung bei elektrisch angetriebenen Rollstühlen, Diplomarbeit, vorgelegt an: Lehrstuhl für Automation der Universität Mannheim, Mannheim, 2003.
- [Bar09] C. Bartolein, A. Wagner, M. Jipp, E. Badreddin (2009). Dependable System Design for Assistance Systems for Electrically Powered Wheelchairs. International Workshop on the Design of Dependable Critical Systems.
- [Bar12] N. Barhoumi, F. Msahli, M. Djemai, K. Busawon Observer design for some classes of uniformly observable nonlinear hybrid systems, Nonlinear Analysis : Hybrid Systems, Volume 6, pp 917-929, 2012.
- [Bas09] F. Basile, P. Chiacchio, and G. De Tommasi. Fault diagnosis and prognosis in petri nets by using a single generalized marking estimation. In 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS), Barcelona, Spain, 2009.
- [Bay08] Mehdi Bayouhd, Louise Travé-Massuyes, Xavier Olive, and Thales Alenia Space. Hybrid systems diagnosis by coupling continuous and discrete event techniques. In Proceedings of the IFAC World Congress, pages 7265–7270, Seoul, Korea, 2008.

- [Bec09] Becker, S.; Koziolok, H.; Reussner, R.: The Palladio component model for model-driven performance prediction. In: *Journal of Systems and Software* 82 (2009), Nr. 1, S. 3 – 22. – ISSN 0164–1212
- [Beh08] BEHRENS, Jan ; GIESECKE, Simon ; JOST, Henning ; MATEVSKA, Jasminka ; SCHREIER, Ulf: *Handbuch der Software- Architektur*. dpunkt.verlag, 2008, S. 33 – 68
- [Bel86] BELLI, F., ECHTLE, K., AND GÖRKE, W., *Methoden und Modelle der Fehlertoleranz*. *Informatik Spektrum* 9, 2 (Apr. 1986), 68-81.
- [Bem99] Bemporad, A. ; Morari, M.: *Control of Systems Integrating Logic, Dynamics, and Constraints*. In: *Automatica* 35 (1999), Nr. 3, S. 407– 427
- [Bil03] Billington, J. et al.: *The Petri Net Markup Language: Concepts, Technology, and Tools*. In: van der Aalst, W.; Best, E. (eds.): *Proc. International Conference Applications and Theory of Petri Nets 2003*. LNCS, Vol.2679, Berlin: Springer, 2003, S. 483-505
- [Bir06] A. Birouche, D. Daafouz, C. Iung, *Observer design for class of discrete time piecewise-linear system*, in: *2nd IFAC Conf. on Analysis and Design of Hybrid Systems*, Alghero, Italy, June, 2006.
- [Bis05] M.Bisiacco and M.E.Valcher. *A behavioral approach to the estimation problem and its applications to state-space models*. submitted for possible presentation at the *43rd IEEE Conference on Decision and Control (CDC)*, Seville (Spain), 2005.
- [Bla03] Blanke, Mogens; Kinnaert, Michel; Lunze, Jan; Staroswiecki, Marcel: *Diagnose und Fault-Tolerant Control*, Springer Verlag, 1. Aufl., Berlin u.a., 2003.
- [Ble05] Bleifuss, Bernd: *Implementierung der Recursive Nested Behaviour Control Structure*, Diplomarbeit vorgelegt an: *Lehrstuhl für Automation der Universität Mannheim*, Mannheim, 2005.
- [Blo03] J. Bloom, C. Clark, C. Clifford, A. Duncan, H. Khan et M. Papantoniou: *Platform Independent Petri net Editor – final report*. *Rap. tech.*, Imperial College, Londres, Royaume-Uni, 2003.
- [Blo04] H. Blom et E. Bloem : *Particle filtering for stochastic hybrid systems*. In *IEEE Conference on Decision and Control*, Atlantis, Les Bahamas, 2004.
- [Bon06a] Bonnet-Torrès, Olivier; Domenech, Patrice; El Bouzidi, Aziz; Lesire, Charles und Tessier, Catherine: *EXHOST-PIPE –PIPE Extended for Two Classes of Monitoring Petri Nets*, in: *27th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*, Turku, Finland, 2006.
- [Bon06b] Bonnet-Torrès, Olivier; Domenech, Patrice; El Bouzidi, Aziz; Lesire, Charles und Tessier, Catherine: *EXHOST-PIPE - Playing Plan Petri Nets and Particle Petri Nets*, in: *27th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*, Tool demonstration at *ATPN'06*, Turku, Finland, 2006.

- [Bou97] Renée Boubour, Claude Jard, Armen Aghasaryan, Eric Fabre, and Albert Benveniste. A petri net approach to fault detection and diagnosis in distributed systems. part i: application to telecommunication networks, motivations, and modelling. In the 36th Conference on Decision & Control, San Diego, California, USA, December 1997.
- [Bou99] A. Bourjij, D. Koenig, An original Petri net state estimation by a reduced Luenberger observer, in: Proceeding of the American Control Conference, San Diego, California, 1999.
- [Bra95] Branicky, M.S.: Studies in Hybrid Systems: Modeling, Analysis, and Control, MIT, Diss., 1995
- [Bre07] Brenner D., Atkinson C., Malak R., Merdes M., Suliman D., Paech B.: Reducing verification effort in component- based software engineering through built-in testing. In: Information Systems Frontiers, vol. 9(2). Springer, Heidelberg (2007)
- [Bro07] Broy, M.; Fox, J.; Hözl, F.; Koss, D.; Kuhrmann, M.; Meisinger, M.; Wild, D.: Service-Oriented Modeling of CoCoME with Focus and AutoFocus. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 177 – 206
- [Bul07] Bulej, L.; Bures, T.; Coupaye, T.; Decky, M.; Jezek, P.; Parizek, P.; Plasil, F.; Poch, T.; Rivierre, N.; Sery, O.; Tuma, P.: CoCoME in Fractal. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 357 – 387
- [Bur07] Bures, T.; Decky, M.; Hnetyuka, P.; Kofron, J.; Patrizek, P.; Plasil, F.; Poch, T.; Sery, O.; Tuma, P.: CoCoME in SOFA. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 388 – 417
- [Cha13] Chantry, E., & Ribot, P. (2013). An integrated frame- work for diagnosis and prognosis of hybrid systems. In 3rd Workshop on Hybrid Autonomous System (HAS). Roma, Italy.
- [Che03] Chen, Zhe: Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond, Technical Report, McMaster University, Canada, 2003.
- [Che07] Chen, Z.; Hannousse, A. H.; Hung, D. V.; Knoll, I.; Li, X.; Liu, Z.; Liu, Y.; Nan, Q.; Okika, J.; Ravn, A. P.; Stolz, V.; Yang, L.; Zhan, N.: Modelling with Relational Calculus of Object and Component Systems - rCOS. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 116 – 145
- [Coc12] Cocquempot, V., El Mezyani, T., & Staroswiecki, M. (2004). Fault detection and isolation for hybrid systems using structured parity residuals. In 5th Asian Control Conference. (Vol. 2, pp. 1204–1212).
- [Crn07] Crnkovic I., Chaudron M., Sentilles S., and Vulgarakis A., “A Classification Framework for Component Models,” in Proceedings of the 7th Conference on Software Engineering and Practice in Sweden, 2007.

- [Dai07] Daigle, M. Roychoudhury, I. Biswas, G. & Koutsoukos, X. (2007). Efficient Simulation of Component- Based Hybrid Models Represented as Hybrid Bond Graphs In Lecture Notes in Computer Science
- [Dav87] R. David, H. Alla: Continuous Petri Nets. In Proc. Of the 8th European Workshop on Application and Theory of Petri Nets, Zaragoza, Spain (1987) 275-294.
- [Dav01] David, René und Alla, Hassane: On Hybrid Petri Nets, in: Discrete Event Dynamic Systems: Theory and Applications, Springer Netherlands, Vol. 11, S. 9-40, Dordrecht, 2001.
- [Dav05] R. David, H. Alla: Discrete, Continuous, and Hybrid Petri Nets. Berlin, Heidelberg, New York: Springer, 2005.–ISBN 3-540-22480-7
- [Dea02] Dearden, R., and Clancy, D. 2002. “Particle filters for real- time fault detection in planetary rovers”. Proceedings of the Thirteenth International Workshop on Principles of Diagnosis. 1-6
- [Dem07] Demchak, B.; Ermagan, V.; Farcas, E.; Huang, T.; Krüger, I. H.; Menarini, M.: A Rich Services Approach to CoCoME. In: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 85 – 115
- [Den04] Denaro G. and M. Pezz`e, “Petri nets and software engineering”, Lectures on Concurrency and Petri Nets, 3098, Springer-Verlag, 2004, 439–466.
- [Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.
- [Dil94] Dill, D. L.; Alur, R.: A Theory of Timed Automata. Theoretical Computer Science 126 (1994) 2, S. 183-235.
- [Dim05] Dimov A, Punnekkat S (2005) On the Estimation of Software Reliability of Component-Based Dependable Distributed Systems. In: Quality of Software Architectures and Software Quality, LNCS 3712:171-187.
- [Dou01] A. Doucet, S. Godsill, C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. Statistics and Computing, 10:197-208, 2001
- [Dou05] Douc, R. and Cappe, O. and Moulines, E., “Comparison of Resampling Schemes for Particle Filtering”, Image and Signal Processing and Analysis, 2005.
- [DRA98] DRATH R., "Hybrid object nets: An object oriented concept for modelling complex hybrid systems". In: [ZAY 98], p. 436-442.
- [Dur06] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. IEEE ROBOTICS AND AUTOMATION MAGAZINE, 2:2006, 2006.
- [Dur12] Durmus M.S., Yildirim U., Soylemez M.T., “Automatic Generation of Petri Net Supervisors for Railway Interlocking Design”, Control Conference (AUCC), 2012 2nd Australian, IEEE, 2012, 180–185.

- [Feh04] A. Fehnker et F. Ivancic: Benchmarks for hybrid systems verification. In International Workshop on Hybrid Systems : Computation and Control (HSCC), Philadelphia, Pennsylvanie, USA, 2004.
- [Fel12] Felderer M., Atkinson C., Barth F. & Breu R. (2012). Model-Driven Testing with Test Sheets In Rech, J., & Bunse, C. (Eds.), Emerging Technologies for the Evolution and Maintenance of software Models (pp. 231-253).
- [Fra94] Frank, P. M.: Diagnoseverfahren in der Automatisierungstechnik. In: Automatisierungstechnik 42 (1994), Nr. 2, S. 47–64
- [Fre98] Frey, G.; Litz L.: Entwurf und formale Verifikation von Steuerungen mit interpretierten Petri-Netzen. Proceedings of the GMA-Kongress Mess- und Automatisierungstechnik "Neue Entwicklungen Technologie Anwendungen", Ludwigsburg, Germany, Jun. 1998; published in: VDI Berichte 1397, VDI-Verlag, pp. 291-298, Düsseldorf, Germany, 1998.
- [Fun04] Funiak, Stanislav: State Estimation of Probabilistic Hybrid Systems with Particle Filters, Master Thesis, Massachusetts Institute of Technology, Cambridge, 2004.
- [Gau14a] Q. Gaudel, E. Chantry, P. Ribot, E. Le Corronc: Hybrid systems Diagnosis using modified particle Petri nets - In Proceedings of the 25th International Workshop on Principles of Diagnosis, September 2014 (DX 2014), Graz, Austria.
- [Gau14b] Q. Gaudel, E. Chantry, P. Ribot: Health Monitoring of Hybrid Systems Using Hybrid Particle Petri Nets - In Proceedings of the Annual Conference of the Prognostics and Health Management Society, September 2014 (PHM 2014), p. 51-61, ISBN 978-1-936263-17-2, Fort Worth, USA.
- [Gau15a] Q. Gaudel, E. Chantry, P. Ribot: Hybrid Particle Petri Nets for Systems Health Monitoring under Uncertainty - International Journal of Prognostics and Health Management (IJPHM), Vol. 6. Special Issue Uncertainty in PHM, June 2015, 022, ISSN 2153-2648.
- [Gau15b] Gaudel, E. Chantry, P. Ribot: Vers une architecture de surveillance de santé d'un système hybride sous incertitudes - Dans 10ème Colloque sur la Modélisation des Systèmes Réactifs, Novembre 2015 (MSR 2015), Nancy, France.
- [Gen07] Sahica Genc and Stéphane Lafortune. Distributed diagnosis of place-bordered petri nets. IEEE Transactions on Automation Science and Engineering, 4(2):206–219, April 2007.
- [Ger98] Gertler, Janos J.: Fault Detection and Diagnostics in Engineering Systems, Marcel Dekker Inc., 1. Aufl., New York, 1998.
- [Goo07] Li P Goodall R Weston P Seng L C Goodman C and Roberts C Estimation of railway vehicle suspension parameters for condition monitoring Control Engineering Practice vol 15 no 1 pp 43–55 Jan 2007
- [Gra07] Grassi, V.; Mirandola, R.; Randazzo, E.; Sabetta, A.: KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability. In: The Common Component Modeling Example (CoCo-Me): Comparing

Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 327 – 356

- [Hal03] P. L. Hall & J. E. Strutt (2003): Probabilistic physics-of-failure models for component reliabilities using Monte Carlo simulation and Weibull analysis: a parametric study. *Reliability Engineering and System Safety* 80, pp. 233–242, doi:10.1016/S0951-8320(03)00032-2.
- [Ham09] F. Hamdi, N. Manamanni, N. Messai and K. Benmahammed, Hybrid observer design for switched linear systems via differentials Petri Nets. *Nonlinear Analysis: Hybrid System*. pp. 310-322. 2009.
- [Har81] Hartmann J. Genrich and Kurt Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science* 13, pages 109–136, 1981.
- [Har87] Harel, D.: Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* 8 (1987) 3, S. 231-274.
- [Har68] P. E. Hart, N. J. Nilsson, B. Raphael, „A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, *IEEE Transactions on Systems Science and Cybernetics SSC4* (2), pp. 100–107, 1968.
- [Hau05] Haug, A. J.: A Tutorial on Bayesian Estimation and Tracking Techniques Applicable to Nonlinear and Non-Gaussian Processes, Mitre Technical Report, Virginia, 2005.
- [Hei01] Heinemann, G. T.; Councill, W. T.: *Component-Based Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2001. – ISBN 0201704854
- [Hen96] Henzinger T. A.: The Theory of Hybrid Automata. In: *Institute of Electrical and Electronics Engineer (IEEE) (Hrsg.): 11th Annual IEEE Symposium on Logic in Computer Science 1996 (LICS '96)*. New Brunswick/USA, 27.-30. Juli 1996. Los Alamitos: IEEE Computer Society Press 1996, S. 278-292.
- [Her08] Herold S., Klus H., Welsch Y., Deiters C., Rausch A., Reussner R., Krogmann K., Koziolk H., Miranda R., Hummel B., Meisinger M., and Pfaller C., “CoCoME: The Common Component Modeling Example,” *The Common Component Modeling Example Comparing Software Component Models*, LNCS 5153, Springer Berlin Heidelberg, pp. 16-53, 2008.
- [Him89] U. Hummert. *Algebraische High-Level Netze*. Dissertation, Technische Universität Berlin (1989)
- [Hof02] Hofbaur, M. W., and Williams, B. C. 2002. “Mode estimation of probabilistic hybrid systems”. *Hybrid Systems: Computation and Control*. 253-266.
- [Hof04] Hofbaur, M. W. and Williams, B. C. 2004. “Hybrid estimation of complex systems”. *Systems, Man, and Cybernetics, Part B: Cybernetics*, *IEEE Transactions on*, 34(5), 2178- 2191
- [Ioa04] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Québec, CANADA H3A 2A7, 2004.

- [Ise84] R. Isermann, "Process fault detection based on modeling and estimation methods—A survey," *Automatica*, vol. 20, no. 4, pp. 387–404, 1984.
- [Ise94] R. Isermann: *Überwachung und Fehlerdiagnose, Moderne Methoden und ihre Anwendungen bei technischen Systemen*, Düsseldorf, VDI-Verlag GmbH, 1994
- [Ise04] Rolf Isermann. Model-based fault detection and diagnosis: status and applications. In *In Proceedings of the 16th IFAC Symposium on Automatic Control in Aerospace*, St, pages 71–85, 2004.
- [Ise06] Rolf Isermann. *Fault-Diagnosis Systems*. Springer, 2006.
- [Jen91] Kurt Jensen. Coloured Petri nets: A high level language for system design and analysis. In Rozenberg, G., editor, *Lecture Notes in Computer Science, Advances in Petri Nets 1990*, volume 483, pages 342–416. Berlin: Springer-Verlag, 1991.
- [Jer06] Jeroen D. Hol, Thomas B. Schön, and Fredrik Gustafsson. On resampling algorithms for particle filters. In *Nonlinear Statistical Signal Processing Workshop*, 2006.
- [Jia13] Wan Jianxiong, Xiang Xudong, Bai Xiaoying, Lin Chuang, Kong Xiangzhen, and Li Jianxiang. Performability analysis of avionics system with multi-layer hm/fm using stochastic petri nets. *Chinese Journal of Aeronautics*, 26(2):363–377, 2013.
- [Jul00] J. Julvez, E. Jimenez, L. Recalde, M. Silva, On observability in timed continuous Petri net system, in: *Proceedings of the First International Conference on the Quantitative Evaluation of System*, IEEE Computer Society, 2000.
- [Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basis Engineering*, 82:35-45, 1960.
- [Kan10] Amr A Kandil, Achim Wagner, Alexander Gotta and Essameddin Badreddin. Collision avoidance in a recursive nested behaviour control structure for Unmanned Aerial Vehicles. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. pp 4276-4281, 2010.
- [Kes92] Kesten, Y.; Pnueli, A.: *Timed and Hybrid Statecharts and Their Textual Representation*. In: Vytupil, J. et al. (Hrsg.): *Formal Techniques in Real-Time and Fault-Tolerant Systemes*. Berlin: Springer 1992, S. 591-619. (Lecture Notes in Computer Science 571)
- [Kou03] X. Koutsoukos, J. Kurien et F. Zhao : Estimation of distributed hybrid systems using particle filtering methods. In *International Workshop on Hybrid Systems : Computation and Control (HSCC)*, Prague, République Tchèque, 2003.
- [Kou09] E. Kourosfar, H. Yaghoubi Shahir, and R. Ramsin, Process Patterns for Component-Based Software Development, In: *Proc. CBSE'09*, 2009, 54-68
- [Kro07] Krogmann, K.; Reussner, R.: Palladio - Prediction of Performance Properties. In: *The Common Component Modeling Example (CoCoMe): Comparing Software Component Models*, Dagstuhl Research Seminar, August 1-3, 2007, 2007, S. 297 – 326

- [KuK07] Kung-Kiu L. and Wang Z., "Software Component Models," IEEE Transactions on Software Engineering, IEEE Computer Society Press, pp. 709-724, 2007.
- [Lap92] Laprie, Jean-Claude: Dependability: Basic Concepts and Terminology, volume 5 of Dependable Computing and Fault-Tolerant Systems. Springer, 1992.
- [LAU08] Copyright© Lehrstuhl für Automation 2008.
- [Lau99] R. Lauber, P. Göhner: Prozessautomatisierung 1, Band 1, 3. Auflage, Berlin Heidelberg, Springer-Verlag, 1999
- [LeB91] J. Le Bail, H. Alla, and R. David, „Hybrid Petri Nets“, Proc. European Control Conference, Grenoble (FR), July 1991.
- [Ler02] Lerner, U., Moses, B., Scott, M., McIlraith, S., Koller, D.: Monitoring a complex physical system using a hybrid dynamic Bayes net. In: UAI'02, Edmonton, AB (2002)
- [Les05] Lesire, Charles und Tessier, Catherine: Particle Petri Nets for Aircraft Procedure Monitoring under Uncertainty, in: ATPN'05, 26 th International Conference On Application and Theory of Petri Nets and Other Models of Concurrency, Miami, Florida, 2005.
- [Les06] Lesire, Charles und Tessier, Catherine: A Hybrid Model for Monitoring and Conflict Prediction in Human Supervised "Autonomous" Systems, in: AAAI Spring Symposium "To boldly go where no human-robot team as gone before", Stanford, CA, USA, 2006.
- [Les07] Lesire, Charles und Tessier, Catherine: Particle Petri Net-Based Estimation in Hybrid Systems to Detect Inconsistencies, in: 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS), 2007, Paris, France.
- [Les88] Leszak, M.; Eggert, H.: Petri-Netz-Methoden Und -Werkzeuge: Hilfsmittel Zur Entwurfsspezifikation Und -Validation Von Rechensystemen, in: Brauer, W: (Hrsg.): Informatik-Fachbereich Nr: 197, Berlin et al: 1988.
- [Lun03] Lunze, J.: Automatisierungstechnik: Methoden für die Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme. Oldenburg Verlag, München, 2003.
- [Luo09] Luo, Y., A. Wagner, L. Mekacher Zouaghi and E. Badreddin, An integrated monitor-diagnosis-reconfiguration scheme for (semi-) autonomous mobile systems. Extended abstract in Workshop on the Design of Dependable Critical Systems, Hamburg, 2009.
- [Lyg03] J. Lygeros, K.H. Johansson, S.N. Simic, J. Zhang, S. Sastry, Dynamical properties of hybrid automata, IEEE Transaction on Automatic Control 48 (1) (2003).
- [Mab12] Y. Mabrouk: „Einsatz von RFID Technologie in der Entwicklung einer Landmarkenbasierten Testumgebung für Roboter Navigation“. Bachelorarbeit am Lehrstuhl für Atomation (Universität Heidelberg) unter Betreuung von Dipl. Ing. Leila Mekacher und Prof. Badreddin (2012).

- [Mat03] Matsuno, H.; Aoshima, H.; Doi, A.; Tanaka, Y.; Matsui, M; Miyano S.: Biopathways Representation and Simulation on Hybrid Functional Petri Net. *Silico Biology* 3 (2003) 3, S. 389-404.
- [Mat10] Matevska, J. (2010), 'Modellbasierte erreichbarkeitsoptimierte Rekonfiguration komponentenbasierter Softwaresysteme zur Laufzeit.', *Softwaretechnik-Trends* 30:1 (2010/02).
- [Mek09a] Mekacher Zouaghi, Koslowski, Alexopoulos, Barth F., Jipp M., Fajardo R., Luo Y., Wagner A. und Badreddin E.: "Dependable component-based design on the Example of a Heating Control System." ,Workshop on the Design of Dependable Critical Systems, Hamburg, 2009
- [Mek09b] Mekacher Zouaghi, Wagner und Badreddin : „Hierarchical hybrid monitoring for autonomous systems.” ,Workshop on the Design of Dependable Critical Systems, Hamburg 2009
- [Mek10] Mekacher Zouaghi, L., A. Wagner and E. Badreddin, Hybrid, recursive, nested monitoring of control systems using Petri nets and particle filters. *Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)*, 73-79, Chicago, June 28 - July 01, 2010.
- [Mek11a] Mekacher Zouaghi, L., A. Alexopoulos, A. Wagner, E. Badreddin, Modified Particle Petri Nets for Hybrid Dynamical Systems Monitoring Under Environmental Uncertainties, *IEEE/SICE International Symposium on System Integration (SII)*, 2011, pp. 497-502.
- [Mek11b] Mekacher Zouaghi L, Alexopoulos A, Wagner A, Badreddin E. Probabilistic Online-Generated Monitoring Models for Mobile Robot Navigation using Modified Petri nets, Tallinn, Estonia, 2011.
- [Mek11c] Mekacher Zouaghi, Wagner, Badreddin, Monitoring of Hybrid Systems using Behavioural System Specification, accepted by 18th IFAC World Congress, Milano, Italy, 2011.
- [Mek12a] Mekacher Zouaghi, L., A. Wagner and E. Badreddin, Benutzerinteraktion beim Online-Monitoring der Navigation eines autonomen Rollstuhls. *Proceedings of AUTOMED-Automatisierungstechnische Verfahren für die Medizin*, Aachen, Germany, March 29-30, 2012.
- [Mek12b] Mekacher Zouaghi, L., A. Alexopolous, M. Koslowski, A. A. Kandil and E. Badreddin, An integrated distributed monitoring for mission-based systems: On the example of an autonomous unmanned helicopter. *6th IEEE International Conference on Intelligent Systems (IS)*, pp. 425-420, Sofia, Bulgaria, 6-8 Sept. 2012.
- [Mek13] Leila Mekacher Zouaghi, Achim Wagner, Essameddin Badreddin: „Design of Fault-Tolerant and Dependable Assistance Systems with Degree of Autonomy Adaptation“ On The International Conference on Advanced Logistics and Transport (ICALT'2013) May 2013, in Sousse, Tunisia.
- [Mek14] Leila Mekacher Zouaghi, Alexander Alexopoulos, Achim Wagner, Essam Badreddin: Mission-based online generation of probabilistic monitoring models for

mobile robot navigation using Petri nets. *Robotics and Autonomous Systems* 62(1): 61-67 (2014)

- [Mic06a] MICROSOFT: DCOM Technical Overview. Version: 2006. <https://msdn.microsoft.com/en-us/library/dd566231.aspx>, Abruf: Apr 26, 2009
- [Mic06b] MICROSYSTEMS, Sun: JSR 220: Enterprise JavaBeans Specification, Version 3.0. <http://java.sun.com/products/ejb/>, Mai 2006
- [Moo99] Moor, T., Raisch, J. 1999. Supervisory control of hybrid systems within a behavioural framework. In Evans, R. J. and Savkin, A. V. (eds.) *Systems and Control Letters*, vol 38, no 3, pp. 157-166
- [NaC04] H. Nait-Charif and S. J. McKenna. Tracking poorly modelled motion using particle filters with iterated likelihood weighting. In *ACCV04*, pages 156-161, 2004
- [Nen01] Nenninger, G.: *Modellbildung und Analyse hybrider dynamischer Systeme als Grundlage für den Entwurf hybrider Steuerungen*. Düsseldorf : VDI-Verlag, 2001
- [Omg06] OMG, Component Model, V4.0, 2006. <http://www.omg.org/technology/documents/formal/components.htm>
- [Pad07] G. D. Padfield, *Helicopter Flight Dynamics: The Theory and Application of Flying Qualities and Simulation Modelling*, Blackwell Publishing Ltd., 2nd edition, 2007.
- [Pao14] Maria Paola Cabasino, Alessandro Giua, and Carla Seatzu. Diagnosability of discrete-event systems using labeled petri nets. *IEEE Transactions on Automation Science and Engineering*, 11(1):144–153, 2014.
- [Pat00] Patton, Ron; Clark, Robert und Frank, Paul M.: *Issues of Fault Diagnose for Dynamic Systems*, Springer Verlag, 1. Aufl., Berlin, 2000.
- [Pay61] Paynter, H. M.: *Analysis and Design of Engineering Systems*. Cambridge, Massachusetts: MIT Press 1961.
- [Pet62] Petri, C. A.: *Kommunikation mit Automaten*. Dissertation Universität Bonn (1962). Bonn: 1962. (Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik an der Universität Bonn 2)
- [Pet90] L., Pete and T. Anderson: *Fault Tolerance - Principles and Practice*. Springer, second, revised edition, 1990.
- [Pet06] S. Pettersson, Designing switched observers for switched systems using multiple Lyapunov functions and dwell-time switching, in: *2nd IFAC Conference on Analysis and Design of Hybrid Systems*, Alghero, Italy. 2006.
- [PNT15] Petri Nets Tools Database Quick Overview (Stand: 2015): <https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html>

- [Ric02] Richard Dearden and Dan Clancy. Particle filters for real-time fault detection in planetary rovers. In In Proceedings of the Thirteenth International Workshop on Principles of Diagnosis, pages 1–6, 2002.
- [Rau08] Rausch, A.; Reussner, R.; Mirandola, R.; Plasil, F.: The Common Component Modeling Example (CoCoMe): Comparing Software Component Models, Dagstuhl Research Seminar, August 1-3, 2007. Bd. 5153. Springer-Verlag, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–85288–9
- [Rei91] W. REISIG: Petri nets and algebraic specifications. – Theoretical Computer Science 80 (1991) 1–34. [ISSN 0304-3975]
- [Rib11] P. Ribot & E. Bensana (2011): A generic adaptative prognostic function for heterogeneous multi-component systems: application to helicopters. In: ESREL 2011, European Safety & Reliability Conference, Troyes, France, doi:10.1201/b11433-53.
- [Rum91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. E. (1991). Object-oriented modeling and design (Vol. 199, No. 1). Englewood Cliffs, NJ: Prentice-hall.
- [RuY09] Yu Ru and Christoforos N. Hadjicostis. Fault diagnosis in discrete event systems modeled by partially observed petri nets. *Discrete Event Dynamic Systems*, 19(4):551–575, 2009.
- [Roy10] Roychoudhury, I., Daigle, M. J. , Biswas, G. & Koutsoukos, X. (2010). Efficient simulation of hybrid systems: A hybrid bond graph approach In Simulation: Transactions of the Society of Modeling and Simulation International
- [Rop12] G. Ropohl: „Allgemeine Systemtheorie: Einführung in transdisziplinäres Denken“. Berlin, edition sigma, 2012 - 246 Seiten.
- [Ros91] Rosenstengel, Bernd und Winand, Udo: Petri-Netze - Eine anwendungsorientierte Einführung, Vieweg & Sohn Verlagsgesellschaft mbH, 4. Aufl., Braunschweig, Wiesbaden, 1991.
- [Rou98] S.I. Roumeliotis, G.S. Sukhatme, and G.A. Bekey. Fault detection and identification in a mobile robot using multiple-model estimation. volume 3, pages 2223–2228. IEEE International Conference on Robotics and Automation, IEEE, May 1998. Leuven, Belgium.
- [Rus04] Russel, P.: Künstliche Intelligenz. Pearson Education Deutschland, 2004
- [Sad13a] Nargess Sadaghzadeh, Leila Mekacher Zouaghi, Achim Wagner, Javad Poshtan, Essameddin Badreddin: “Cascaded Error Estimation and Compensation with Optical and Inertial Sensors Fusion in Robotic Surgery“. On the competitiveness through Maintenance and Assest Management (COMADEM 2013), June 2013, in Helsinki, Finland
- [Sad13b] Nargess Sadeghzadeh Nokhodberiz, Leila Mekacher Zouaghi, Achim Wagner, Eugen Nordheimer, Javad Poshtan and Essameddin Badreddin. Cascaded Kalman Filters for Error Estimation and Compensation of Inertial Navigation System

Fusing Optical and Inertial Sensors. In Proceedings of the 5th International Conference on Mechatronics (ICOM 2013). Juli 2013.

- [Sam05] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneket- zis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.
- [San08] Andre M. Santana, Anderson A. S. Souza, Ricardo S. Britto, Pablo J. Alsina, and Adelardo A. D. Medeiros. Localization of a mobile robot based in odometry and natural landmarks using extended kalman filter. In *ICINCO-RA (2)*, pages 187–193, 2008.
- [Sch01] Schnabel, M.: *Diskret-Kontinuierliche Dynamische Systeme: Opti- male Steuerung und Beobachtung*. Düsseldorf, Germany : VDI-Verlag, 2001
- [Sco09] Scout B1-100 UAV©Aeroscout GmbH Schwtizerland
- [Set02] Vachirasuk Setalaphruk, Takashi Uneno, Yasuyuki Kono, and Masatsugu Kidode. Topological map generation from simplified map for mobile robot navigation. In *The 16th Annual Conference of Japanese Society for Artificial Intelligence*, 2002.
- [Sim03] Simani, Silvio; Fantuzzi, Cesare und Patton, Ron J.: *Model-based Fault Diagnose in Dynamic Systems Using Identification Techniques*, Springer Verlag, 1. Aufl., London, 2003.
- [Sri93] Srinivasan, V. S. und Jafari, Mohsen A.: *Fault Detection/Monitoring Using Time Petri Nets*, *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, No. 4, 1993.
- [Sun15] B. Esther Sunanda, P. Seetharamaiah, “Modeling of Safety-Critical Systems Using Petri Nets”, *SIGSOFT Softw. Eng. Notes*, 40:1 (2015), 1–7.
- [Sto96] Storey, Neil: *Safety-critical Computer Systems*. Addison-Wesley, 1996.
- [Tur04] TURAU, Volker: *Algorithmische Graphentheorie*. Oldenbourg Wissenschaftsverlag, 2004. – ISBN 3–486–20038–0
- [Thr05] Thrun, S. ; Burgard, W. ; Fox, D. ; Arkin, Ronald C. (Hrsg.): *Probabilistic Robotics*. Bd. 1. III. The MIT Press, 2005
- [Val99] M.E. Valcher and J.C. Willems. Observer synthesis in the behavioral approach. *IEEE Trans. on Aut. Contr.*, 44, no.12:2297–2307, 1999.
- [Vee04] H. Veeraraghavan et N. Papanikolopoulos : Combining multiple tracking modalities for vehicle tracking in traffic intersections. In *IEEE International Conference on Robotics and Automation (ICRA)*, La Nouvelle-Orléans, Louisiane, USA, 2004.
- [Ven03a] V. Venkatasubramanian. A review of process fault detection and diagnosis Part I: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293–311, March 2003.

- [Ven03b] V. Venkatasubramanian. A review of process fault detection and diagnosis Part II: Qualitative models and search strategies. *Computers & Chemical Engineering*, 27(3):313–326, March 2003.
- [Ver04] Verma, Vandi: Tractable Particle Filters for Robot Fault Diagnose, Robotics Institute, Carnegie Mellon University, Pittsburgh, 2004.
- [Wag06] Wagner, T.: Qualitative sicht-basierte Navigation in unstrukturierten Umgebungen, Universität Bremen, Dissertation, 2006
- [Wag10] Achim Wagner, Meike Jipp, Amr A Kandil, Christoph Eck and Essameddin Badreddin. Generic system architecture for dependable interactive systems: A flying robot example. In *Systems Man and Cybernetics (SMC)*, 2010 IEEE International Conference on. pp 2129-2136, 2010.
- [Wag13] Achim Wagner, Leila Mekacher Zouaghi, Florian Barth, Colin Atkinson, and Essameddin Badreddin: „ Behavior-Based Built-in Testing and Monitoring for Dependability Assessment of Dynamical Systems“. On the competitiveness through maintenance and Assesst Management (COMADEM 2013), June 2013, in Helsinki, Finland
- [Wel01] G. Welch and G. Bishop. An introduction to the Kalman filter. Course 8. In *Proceedings of the ACM SIGGRAPH Conference*, Los Angeles, August 2001. ACM Press.
- [W3C15] Extensible Markup Language. <http://www.w3.org/XML>
- [Wil89] J.C. Willems, Models for dynamics. *Dynamics Reported Volume 2*, pages 171-269, 1989
- [Wil91] J.C. Willems, Paradigms and puzzles in the theory of dynamical systems. *IEEE Transactions on Automatic Control*. Volume 36, pages 259-294, 1991
- [Wil97] J.C. Willems, On interconnections, control, and feedback. *IEEE Transactions on Automatic Control Volume 42*, pages 326-339, 1997
- [Wil02] J.C. Willems: State construction in discrete event and continuous systems. *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, Nevada, pages 13-15, 2002
- [Yan03] YANG, S.: A Condition-Based Failure-Prediction and Processing-Scheme for Preventive Maintenance. *IEEE Transactions on Reliability* 52 (2003) 3, S. 373–383.
- [Zha01] F. Zhao, X. Koutsoukos, H. Haussecker, J. Reich et P. Cheung : Distributed monitoring of hybrid systems : a model-directed approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, Washington, USA, 2001.
