Modellierung von Steuerungssystemen mit Signal-Petrinetzen Eine Fallstudie aus der Automobilindustrie

Jörg Desel, Gabriel Juhás, Robert Lorenz, Vesna Milijic, Christian Neumair
Lehrstuhl für Angewandte Informatik
Katholische Universität Eichstätt–Ingolstadt
Ostenstr. 28, 85071 Eichstätt
E-Mail: joerg.desel@ku-eichstaett.de

Reinhard Schieber
Audi AG, I/EE 33
85045 Ingolstadt

Abstract: In dem DFG-geförderten Projekt SPECIMEN werden Verfahren zur Modellsynthese und -validierung im Steuerungsentwurf entwickelt. Allgemeines Ziel des Projektes ist, aus einer Beschreibung der Strecke und aus verschiedenen semi-formal gegebenen Anforderungen schrittweise ein valides Modell zu erzeugen, das anschliessend als formale Spezifikation der Steuerung und als Testreferenz dienen kann. Schwerpunkt des hier vorgestellten Ansatzes ist die werkzeuggestützte Validierung von formalisierten Anforderungen durch Simulation, d.h. durch Generierung und Visualisierung kausaler Abläufe. Als Modellierungssprache werden durch Signalkanten erweiterte Petrinetze verwendet. Im Rahmen eines Industrieprojektes wurden die SPECIMEN-Konzepte für ein aktuelles Problem der Fahrzeugindustrie angewandt und erweitert. Diese Fallstudie wird in der vorliegenden Arbeit von den Anforderungen bis zu den detaillierten Modellen vorgestellt. Es wird über Erfahrungen aus diesem Projekt berichtet, insbesondere zu Übereinstimmungen und Differenzen zwischen den in SPECIMEN getroffenen Annahmen und der in dieser Industriesparte vorgefundenen Realität.

Stichworte: Petrinetze, Modellsynthese, Validierung, kausale Abläufe

1 Einleitung

Die Verwendung von Modellen bei der Software-Entwicklung im allgemeinen erhält in Forschung und Praxis eine immer größere Bedeutung, nicht zuletzt durch die Verbreitung der in der UML (Unified Modeling Language) zusammengefassten Modellierungssprachen und durch die Standardisierung modellbasierter Entwicklungsprozesse. Die Entwicklung von Software für Eingebettete Systeme dagegen basiert immer noch weitgehend auf ad-hoc Implementierungen informeller Spe-

zifikationen, die typischerweise von Ingenieuren verschiedener Teildisziplinen auf unterschiedlichem Abstraktionsgrad erstellt werden. Dies mag zum Teil durch die geringere Komplexität der einzelnen Softwaremodule in Eingebetteten Systemen begründet sein, oft führt die Implementierung durch den Fachexperten mit vielen (durch denselben Fachexperten durchgeführten) anschliessenden Tests und Korrekturen zu einer zufriedenstellenden Lösung. Ein Hinderungsgrund für die Verwendung modellbasierter Entwicklungsverfahren für komplexe Systeme mit vielen eingebetteten Software-Modulen ist das Fehlen einer geeigneten Modellierungssprache für alle beteiligten Ingenieure und Software-Entwickler. Diese Situation ist aus wenigstens den folgenden zwei Gründen sehr unbefriedigend:

- Eingebettete Systeme sind häufig sicherheitskritische Systeme. Jedes moderne Kraftfahrzeug ist Beispiel eines komplexen Systems mit vielen miteinander vernetzten Software-Komponenten. Eine Ad-hoc Implementierung einzelner Steuerungskomponenten kann trotz durchgeführter lokaler Tests zu Fehlfunktionen führen, die katastrophale Konsequenzen haben können (z.B. unbeabsichtigte Fehlfunktionen von Airbag, Bremsen, Lenkung u.s.w.).

- Fachwissen, Software-Implementierung, Testfallgenerierung und Testbewertung liegen nicht in einer Hand, sondern bei mehreren Personen unterschiedlicher Ausbildung in getrennten Unternehmen. So werden in der Automobilindustrie Anforderungen von Entwicklungsingenieuren verschiedener Abteilungen in unterschiedlicher (meist informeller) Weise formuliert. Diese Anforderungsdokumente sind in Unternehmen der Fahrzeugelektronik Grundlage für den Entwurf von Kontroll-Software für spezielle Hardware-Bausteine, und sie sind anschliessend Grundlage für den Test der Software durch spezialisierte Testingenieure. Unvollständigkeiten, implizite Annahmen oder Missverständnisse in den Anforderungsdokumenten führen zu fehlerhaften Modulen, deren Korrektur zeit- und kostenaufwändig ist. Zudem ist eine Fehlererkennung in der Testphase keineswegs garantiert, denn der Testingenieur kann auch nur auf die oft unzureichenden Anforderungsdokumente zugreifen (abgesehen davon, dass Tests ohnehin nur Fehler aufdecken, aber nicht Fehlerfreiheit beweisen können).

Dem genannten Problem widmet sich das durch die DFG geförderte Projekt SPECIMEN[1], ein Teilprojekt des Schwerpunktprogramms *Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen*. Die grundsätzliche Zielsetzung dieses Projektes ist die Entwicklung von Konzepten und Verfahren zur Synthese von Modellen aus verschiedenen Spezifikationen, die teilweise informal und auf unterschiedlichen Abstraktionsebenen formuliert sind. Diese Modelle umfassen jeweils sowohl das zu steuernde System (die Strecke) als auch den Kontrollalgorithmus. Ihr Verhalten soll also das Verhalten des Gesamtsystems widerspiegeln. Neben formalen Aspekten spielt die praktische Machbarkeit in dem Projekt eine wichtige Rolle. Die Mo-

[1] Integration von Spezifikations- und Modellierungstechniken bei der Modellsynthese im Steuerungsentwurf. Projektleiter: Prof. Dr. Jörg Desel, Angewandte Informatik, KU Eichstätt-Ingolstadt und Prof. Dr. H.-M. Hanisch, Automatisierungstechnik, Uni Halle-Wittenberg

dellsynthese kann naturgemäß nicht vollautomatisch geschehen, viele Schritte bedürfen der Validierung durch Rückkopplung mit dem Fachexperten [De00, De02]. Insbesondere müssen die verwendeten Sprachen, Grafiken und Visualisierungen so gestaltet sein, dass die einschlägigen Fachexperten und andere Beteiligte die Modelle inhaltlich nachvollziehen können. Grundlage des verwendeten Modells sind modulare Petrinetze, die durch Signalaustausch miteinander kommunizieren [HaLu00, JuLo02, St00]. Diese Modelle sind einerseits hinreichend formal um Analyse- und Simulations-Verfahren einsetzen zu können, die wiederum Grundlage der Validierung sind. Andererseits bieten Petrinetze eine anschauliche Darstellung, die ggfs. durch Verwendung anderer Symbole an konkrete Umgebungen angepasst werden kann [Schn99].

Ein valides Modell kann an wenigstens zwei Stellen nutzbringend eingesetzt werden. Es dient erstens als vollständige und präzise Spezifikation der zu entwickelnden Software-Module. Nur wenn diese, eingebettet in das als bekannt vorausgesetzte ungesteuerte System, dasselbe Verhalten wie sein Modell zeigen, handelt es sich um eine korrekte Implementierung. Zweitens nützt das Modell in der Testphase. Das fertige System und sein Modell können parallel getestet werden, bei Ergebnisabweichungen wird ein Fehlerfall diagnostiziert. Noch wichtiger als der Einsatz des fertigen *Modells* sind Ergebnisse während des *Modellierens*. Defizite und Unklarheiten eines Anforderungsdokumentes werden oftmals beim Versuch seiner Formalisierung deutlich. Fehlende oder unpassende Schnittstellen zwischen verschiedenen Modulen werden beim Modellieren aufgedeckt. Redundante Anforderungen lassen sich erkennen und reduzieren. Widersprüchliche Anforderungen werden durch die Nichtexistenz eines korrekten Modells aufgedeckt.

Die Modellierungsphase besteht nicht einfach aus der Übertragung aller Anforderungen in ein geeignetes mathematisches Modell. Vielmehr kann diese Modellierung nur mehrstufig geschehen. Beginnend mit einfachen Modellen werden iterativ neue Anforderungen formuliert, (halb-) automatisch integriert und das Ergebnis validiert [HaLu99, De00, JuLo02, De02]. Ausgangspunkt sind stets ein Modell der ungesteuerten Strecke, vorgegebene Eigenschaften sowie Szenarien, also gewünschte Abläufe des gesteuerten Systems.

Diese Arbeit stellt eine industrielle Fallstudie im Rahmen eines Pilotprojekts mit der Audi AG vor, in der die in SPECIMEN entwickelten Konzepte auf ein reales System mit aktuellen Fragestellungen angewandt wurden. Dazu mussten einige SPECIMEN-Konzepte für den speziellen Anwendungsbereich leicht angepasst werden. Es war das Ziel dieser Fallstudie, die tatsächliche Problemstellung soweit möglich abzubilden und sie nicht auf den in SPECIMEN entwickelten Formalismus einzugrenzen.

Im folgenden, zweiten Abschnitt wird die Fallstudie - Modellierung von Steuerungskomponenten für die Tank- und Resttankanzeige in Kraftfahrzeugen - skizziert. Der dritte Abschnitt stellt das Vorgehensmodell dar.

Als formale Modellierungssprache wurden im Projekt SPECIMEN Signal-Petrinetze [HaLu00, St00], [DeJuLo00], [JuLo02, DeJuLo02] ausgewählt. Es hat sich allerdings herausgestellt, dass die im Projekt vorgesehene Aufteilung der Spezifikation in ein Modell der Strecke und in zusätzliche zu implementierende Anforderungen in diesem Anwendungskontext nicht überzeugt. Vielmehr

müssen Ansätze zu Modularitätsprinzpien, zur eingeschränkter Interaktion zwischen Modulen und zur konsequenten Unterscheidung kontrollierbarer, beobachtbarer und interner Aktionen eingesetzt werden. Erweiterungen betreffen ein Zeitkonzept zur Darstellung von Realzeit-Bedingungen, die Berücksichtigung von reellwertigen Sensordaten durch Integration von Konzepten höherer Petrinetze sowie eine anwendergerechtere Darstellungsform. Bis auf den letzten Punkt wurden die Erweiterungen durchgeführt. Der Formalismus wird in Abschnitt vier dargestellt.

Die Abschnitte fünf bis sieben sind den konkreten Modellen gewidmet. Es wird jeweils das Modell eines Teilalgorithmus vorgestellt, aus der Vorgabe entwickelte Eigenschaften werden formuliert und diese schließlich exemplarisch verifiziert. Die Aufteilung in Teilmodelle erfolgte unter darstellungstechnischen Aspekten und hat sich aufgrund der Vorgabe angeboten. Schließlich hat sich aber gezeigt, dass eine Aufteilung nach Modularitätsprinzipien etwas anders verlaufen wäre.

Der achte Abschnitt geht auf die verfügbaren Werkzeuge und auf notwendige Erweiterungen ein. Der neunte Abschnitt enthält zusammenfassend gewonnene Erfahrungen aus der Fallstudie.

2 Beschreibung der Fallstudie

Im Rahmen der Entwicklung einer neuen Automobilserie der Audi AG behandelt die Fallstudie die Steuerung der Tank- und Resttankanzeige. Überraschend mag sein, dass diese Thematik ausgesprochen komplex ist. So erfolgt die Bestimmung des aktuellen Tankinhalts in einigen Fällen mit Hilfe verschiedener Sensoren, in anderen jedoch durch eine Berechnung aus dem Verbrauch und einem früheren Tankinhaltswert. Dabei spielen verschiedene Parameter (Zündung ein/aus, Fahrzeug steht/fährt, Motor ein/aus) eine wichtige Rolle. Zudem wird die Schräglage des Fahrzeugs durch Sensoren erfasst. Aufgrund der speziellen Tankform tragen - abhängig vom momentanen Füllstand - nur einige der Sensoren zur Bestimmung des Inhalts bei. Fällt ein Sensor aus, so führt ein Plausibilitätstest dazu, dass dessen Sensordaten nicht mehr in die Berechnung eingehen. Die verbrauchsbasierte Berechnung ist zwar sehr genau, aber kleine Fehler unterhalb einer Toleranzschwelle können sich zu einem signifikanten Unterschied zwischen tatsächlichem und berechnetem Tankwert summieren. Diese Ausführungen zeigen bereits, dass zur Steuerung dieses technischen Systems (das kontinuierliche und diskrete Elemente enthält) komplexe Algorithmen nötig sind.

Ausgangspunkt der Fallstudie war eine informelle Beschreibung des obigen Sachverhalts. Daraus ergaben sich folgende Aufgaben:

- Anpassung geeigneter Modellierungstechniken und Modellsynthesetechniken,
- Entwicklung von Modellen für die Steuerung der Tankfunktionen eines Fahrzeuges,
- Simulation und Validierung der Modelle,
- Untersuchung der Machbarkeit vergleichbarer Aufgabenstellungen im größeren Maßstab,
- Präzisierung der Anforderungen an später zu entwickelnde Software-Werkzeuge.

3 Vorgehensmodell

Der Ausgangspunkt sollte zu Beginn des Projekts ein Grundmodell (des ungesteuerten Systems) und eine Menge von (informell oder formal) angegebenen Anforderungen sein, die durch systematische Modellerweiterungen sicherzustellen sind [HaLuRa97, HaThLu97]. Tatsächlich aber haben sich die Vorgaben der Audi AG gänzlich anders dargestellt: Gegeben waren natürlichsprachliche Beschreibungen von Szenarien, die durch den zu modellierenden Algorithmus realisiert werden müssen und darin verwoben wenige recht implizit formulierte Anforderungen. Das Grundmodell der technischen Anlage wird teilweise durch Erläuterungen der entsprechenden Fahrzeugkomponenten gegeben (z.B. Funktionsweise der vier Geber im Tank) und teilweise als bekannt vorausgesetzt (z.B. mögliche Zustände der Zündung und der Bewegung des Fahrzeugs). Einige dieser impliziten Annahmen konnten leicht ergänzt werden, für andere waren Rückfragen bei der Audi AG notwendig. Bei der Interpretation der Szenarienbeschreibungen wurden Mehrdeutigkeiten entdeckt, die zum Teil auch innerhalb der Audi AG nicht sofort aufgeklärt werden konnten. Insgesamt hat sich die nicht sehr überraschende Annahme bestätigt, dass die Modellbildung in hohem Maße von Rücksprachen mit Fachexperten abhängt und nicht allein aufgrund der gegebenen Dokumente erfolgen kann. Daher ist den ersten Schritten bei der Modellierung und insbesondere der wiederholten Validierung im Vorgehensmodell ein besonderer Stellenwert zuzumessen [De00, De02].
Der vorgesehene Vorgehensmodell wird im folgenden dargestellt. Die späteren Phasen der modellbasierten Testfallgenerierung konnten in dem kleinen Anwendungsbeispiel nur exemplarisch dargestellt werden.

Vorgehensmodell

Die ersten sechs Schritte beziehen sich auf einzelne Module

1. Extraktion und Validierung eines Streckenmodells.
2. Extraktion, Formulierung und Validierung von Anforderungen.
3. Modellierung von Abläufen bzw. Ablaufschemata aus den Szenarien, unter Einbeziehung des Streckenmodells.
4. Generierung des Gesamtsystems durch Faltung der Abläufe. Dieses Gesamtsystem unterstützt automatisch die Ausgangsabläufe, kann aber noch weitere (gewünschte und unerwünschte) Abläufe besitzen.
5. Ausschluss unerwünschter Abläufe durch Implementierung der in 2. gewonnenen Anforderungen. Dabei muss gewährleistet werden, dass die vorgegebenen Abläufe aus 3. noch möglich sind.
6. Validierung der einzelnen Module und Verifikation der Anforderungen.
7. Integration der Module durch Komposition über festgelegte restriktive Schnittstellen.
8. Verifikation des Gesamtsystems.
9. Einsatz als Referenzmodell.
10. Analyse zur effizienten Generierung relevanter Testdaten.
11. Einsatz als Testreferenz durch Simulator, parallel zur realisierten Steuerung.

4 Beschreibung der Modellierungssprache

Signal-Petrinetze (vgl. [SrKr91, HaLu00, St00, JuLo02]) sind eine Erweiterung klassischer Petrinetze. Ein Signal-Petrinetz ist ähnlich wie ein klassisches Petrinetz (vgl. [DeRe98]) ein Graph mit zwei verschiedenen Sorten von Knoten.

Stellen Eine Knotensorte sind Stellen. Stellen werden graphisch durch Kreise dargestellt und können Marken enthalten. Jeder Stelle ist eine Menge von Markenarten (genannt Bereich) zugeordnet. Grundsätzlich können mehrere Marken gleicher oder auch verschiedener Art auf einer Stelle liegen. In unserer graphischen Repräsentation werden Ausgabestellen, wie beispielsweise die aktuelle Tankinhaltsanzeige, grau unterlegt. Wir unterscheiden zwei verschiedene Sorten von Stellen, nämlich Low-Level- und High-Level-Stellen.
Jede *Low-Level-Stelle* wird graphisch durch einen einfachen Kreis dargestellt. Der Bereich von Low-Level-Stellen hat genau eine Markenart, eine schwarze Marke. Die Anzahl der schwarzen Marken auf der Stelle repräsentiert den Zustand der Stelle. In den im Projekt verwendeten Netzen symbolisieren die Low-Level-Stellen Bedingungen und enthalten höchstens eine schwarze Marke, nämlich falls die Bedingung erfüllt ist und keine Marke, falls sie nicht erfüllt ist.
Jede *High-Level-Stelle* wird graphisch durch zwei konzentrische Kreise dargestellt. Im Unterschied zu Low-Level-Stellen ist der Bereich einer High-Level-Stelle eine beliebige nichtleere Menge, so dass unterschiedliche Marken vorliegen können. Der Zustand einer High-Level-Stelle ist durch die Anzahl der Marken jeder Markenart festgelegt. In den präsentierten Beispielen ist der Bereich von High-Level-Stellen stets die Menge der reellen Zahlen. Zudem enthält jede Stelle höchstens eine Marke. Durch diese Beschränkung wird der Zustand einer High-Level-Stelle einfach durch eine reelle Zahl dargestellt. Sie repräsentiert beispielsweise den Füllstand des Tanks.

Transitionen Die andere Sorte von Knoten sind Transitionen. Sie werden graphisch durch Rechtecke dargestellt und repräsentieren Aktionen oder Operationen. In unserer graphischen Repräsentation werden Ereignisse, die nicht steuerbar sind, durch grau unterlegte Transitionen dargestellt. Dies sind typischerweise Benutzeraktionen (z.B. Einschalten der Zündung) oder Fehler (z.B. Geberkurzschluss). Transitionen, die hellgrau unterlegt sind, symbolisieren Aktionen, die neue Werte generieren, wie beispielsweise die Berechnung von Mittelwerten aus den Sensordaten.

Kanten Knoten in Signalnetzen können durch verschiedene Sorten von Kanten verbunden sein.
Flusskanten sind schwarze Pfeile. Sie verbinden entweder Stellen mit Transitionen oder Transitionen mit Stellen. Die Flusskanten können eine Zeitbeschriftung haben. Zudem sind Flusskanten zwischen High-Level-Stellen und Transitionen durch eine Variable beschriftet.
Testkanten sind schwarze Doppelpfeile zwischen Transitionen und Stellen. Testkanten zwischen High-Level-Stellen und Transitionen sind durch eine Variable beschriftet.

Überschreibungskanten sind graue Doppelpfeile zwischen Transitionen und High-Level-Stellen. Sie sind mit zwei Variablen versehen und können zusätzlich eine Zeitbeschriftung besitzen.
Synchronisationskanten sind gezackte Pfeile zwischen Transitionen.

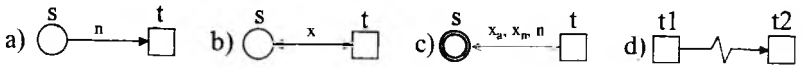

Abbildung 1: a) Flusskante, b) Testkante, c) Überschreibungskante, d) Synchronisationskante

Das *dynamische Verhalten* von Signalnetzen wird durch das Schalten von Transitionen bestimmt. Ob eine Transition schalten kann und wie sie die Markierung bestimmter Stellen beim Schalten beeinflusst, wird durch die umgebenden Kanten bestimmt.
Transitionen, die nicht mit High-Level-Stellen verbunden sind (sogenannte *Low-Level-Transitionen*), können schalten, falls in jeder Low-Level-Stelle, von der eine Flusskante oder eine Testkante zu der Transition führt, eine Marke liegt.

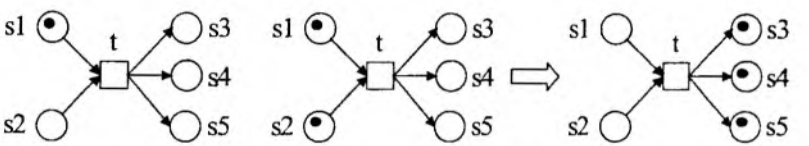

Abbildung 2: Links ein Beispiel eines Netzes in einem Zustand, in dem die Low-Level-Transition t nicht schalten kann, weil die Stelle s_2 keine Marke enthält. In der Mitte das gleiche Netz in einem Zustand, in dem t schalten kann. Rechts das Netz im Zustand nach dem Schalten von t.

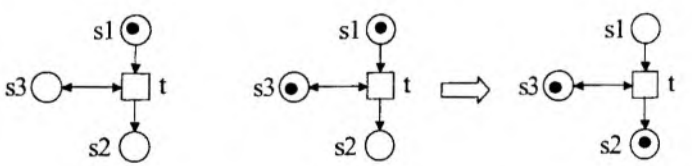

Abbildung 3: Links ein Beispiel eines Netzes mit einer Testkante in einem Zustand, in dem die Low-Level-Transition t nicht schalten kann, weil die Stelle s_3 keine Marke enthält. In der Mitte das gleiche Netz in einem Zustand, in dem t schalten kann. Rechts das Netz im Zustand nach dem Schalten von t.

Jede Variable einer Kantenbeschriftung kann durch einen Wert aus dem Bereich der anliegenden High-Level-Stelle substituiert werden. In unserem Fall bedeutet dies, dass die Variablen durch reelle Zahlen ersetzt werden. Eine Transition, die mit High-Level-Stellen durch Kanten verbunden ist (*High-Level-Transition*), kann eine *Schaltbedingung* haben. Die Schaltbedingung ist ein logischer Ausdruck, der die Variablen aus den Kantenbeschriftungen beinhaltet. Zum Schalten einer

High-Level-Transition werden Werte für die Variablen an den umgebenden Kanten substituiert, so dass die Variable einer Kante (bei Überschreibungskanten die erste Variable), die von einer High-Level-Stelle zu der Transition führt, durch eine Marke der Stelle substituiert wird, und die Schaltbedingung für die Transition unter den substituierten Werten erfüllt ist. Zudem müssen die Vorausetzungen für das Schalten von Low-Level-Transitionen für die Low-Level-Stellen, von denen eine Kante zu der High-Level-Transition führt, erfüllt sein.

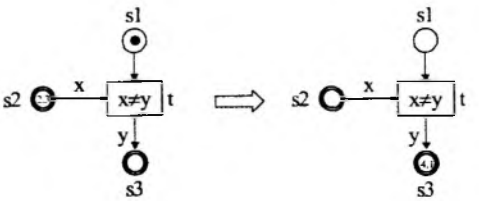

Abbildung 4: Dieses Netz besitzt zwei High-Level-Stellen (s_2 und s_3). Die Stelle s_2 enthält die Marke 2,3. Zum Schalten der Transition werden die Variablen x und y substituiert. Die Transition kann für die Substitution $x = 4{,}1$ nicht schalten, weil der Wert von x nur durch die Marke 2,3 substituiert werden kann. Die Transition kann auch für $x = 2{,}3$ und $y = 2{,}3$ nicht schalten, da die Schaltbedingung $x \neq y$ nicht erfüllt ist. Sie kann aber für $x = 2{,}3$ und $y = 4{,}1$ schalten, weil fr diese Substitution $x \neq y$ gilt. Rechts das Netz im Zustand nach dem Schalten von t für $x = 2{,}3$ und $y = 4{,}1$.

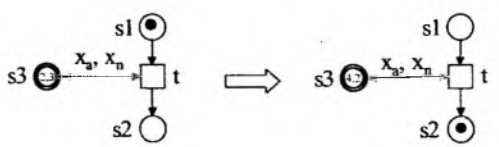

Abbildung 5: Links ein Netz mit einer Überschreibungskante von s_3 nach t. Rechts das Netz im Zustand nach dem Schalten von t für die Substitution $x_a = 2{,}3$ und $x_n = 4{,}2$.

Das Schalten einer Transition (Low- oder High-Level-) löscht eine Marke aus jeder Low-Level-Stelle, von der eine Flusskante zu der Transition führt und fügt eine Marke zu jeder Low-Level-Stelle hinzu, zu der eine Flusskante von der Transition führt.
Zudem löscht das Schalten einer High-Level-Transition aus jeder High-Level-Stelle, von der eine Flusskante zu der Transition führt, den aktuellen Wert und fügt zu jeder High-Level-Stelle, zu der eine Flusskante führt, die für die Variable der entsprechenden Kante substituierte Marke hinzu, und löscht aus jeder High-Level-Stelle, die mit der Transition durch eine Überschreibungskante verbunden ist, den aktuellen Wert und fügt dort den in der zweiten Variablen der Überschreibungskante substituierten Wert ein. Führt zusätzlich von einer Transition, die schalten kann, eine Synchronisationskante zu einer anderen Transition, die ebenfalls schalten kann, so wird beim Schalten der

ersten Transition die zweite Transition gleichzeitig mitschalten, d. h. die zweite Transition wird von der ersten synchronisiert. Eine Transition, zu der eine Synchronisationskante führt, wird nie schalten, ohne von einer Transition synchronisiert zu sein.

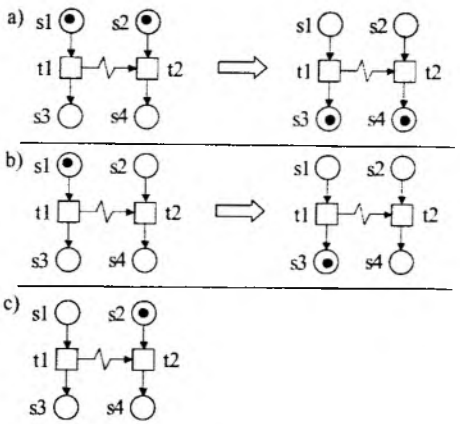

Abbildung 6: a) Links ein Netz mit einer Synchronisationskante von t_1 nach t_2, in dem beide Transitionen schalten können. Durch das Schalten von t_1 wird t_2 synchronisiert, so dass die Transition t_2 mitschaltet. Rechts das Netz nach dem Schalten von t_1 und t_2.
b) Links das obige Netz, in dem nur die Transition t_1 schalten kann. Da t_2 nicht schalten kann, wird sie von t_1 nicht synchronisiert. Rechts das Netz nach dem Schalten von t_1.
c) Das obige Netz, in dem die Transition t_1 nicht schalten kann. Die Transition t_2 wird nicht schalten, da sie von t_1 nicht synchronisiert wird.

Eine *Zeitbeschriftung* an einer Kante von einer Stelle zu einer Transition bewirkt: Falls die Transition schalten kann, dann schaltet sie unmittelbar, wenn die Zeit der Zeitbeschriftung einer jeden Kante, die eine Zeitbeschriftung hat und zu der Transition führt, abgelaufen ist. Eine Zeitbeschriftung an einer Kante von einer Transition zu einer Stelle wird wie folgt interpretiert: Das Schalten der Transition fügt nach der Zeit der Zeitbeschriftung die entsprechende Marke in der Stelle hinzu. Die Zeitbeschriftung bei der Überschreibungskante zwischen einer Stelle und einer Transition bedeutet, dass beim Schalten der Transition nach Ablauf der Zeit der Zeitbeschriftung der alte Wert durch den neuen Wert ersetzt wird.
Steuerbare Transitionen, zu denen nur Flusskanten ohne Zeitbeschriftung führen, und die auch nur durch Überschreibungskanten ohne Zeitbeschriftung mit Stellen verbunden sind, unterliegen der *Progressannahme*. Diese besagt, dass von einer Menge solcher paarweise in Konflikt stehender Transitionen auch eine schalten wird.
Hierbei stehen zwei Transitionen, die beide schalten können, in Konflikt miteinander, wenn nach dem Schalten der einen Transitionen die andere Transitionen nicht mehr schalten kann.

5 Modell der Nachtankerkennung bei *Zündung aus*

In den Kapiteln fünf bis sieben wird das Fallbeispiel präsentiert. Aus Platzgründen können nicht alle Details behandelt werden. Am Anfang der einzelnen Kapitel steht jeweils eine kurze Beschreibung der gegebenen Spezifikation des entsprechenden Moduls. Anschließend folgt das Signal-Petrinetz und eine Beschreibung des Verhaltens, veranschaulicht durch einzelne Abläufe. In den Kapitel fünf und sechs werden an Beispielen Anforderungen verifiziert, in Kapitel sieben wird auf die Generierung von Testfällen eingegangen.

Die Nachtankerkennung bei *Zündung aus* besteht aus zwei Messphasen. Die erste Messphase wird durch das Ausschalten der Zündung gestartet und liefert den Mittelwert *MWalt*, das Einschalten der Zündung leitet die zweite Messphase ein, die den Mittelwert *MWneu* liefert. Falls die Differenz der beiden Mittelwerte größer als 4 Liter ist, wird auf Nachtanken erkannt und der Tankinhalt neu berechnet (durch Addition der Differenz).

Für die Nachtankerkennung sind folgende Anforderungen zu erfüllen:

(i) Reicht die Zeit, während der die Zündung aus ist, nicht aus, um Messphase 1 zu durchlaufen, wird keine Nachtankerkennung durchgeführt.

(ii) Das Ergebnis von Messphase 1 überlebt kurzfristiges Einschalten der Zündung.

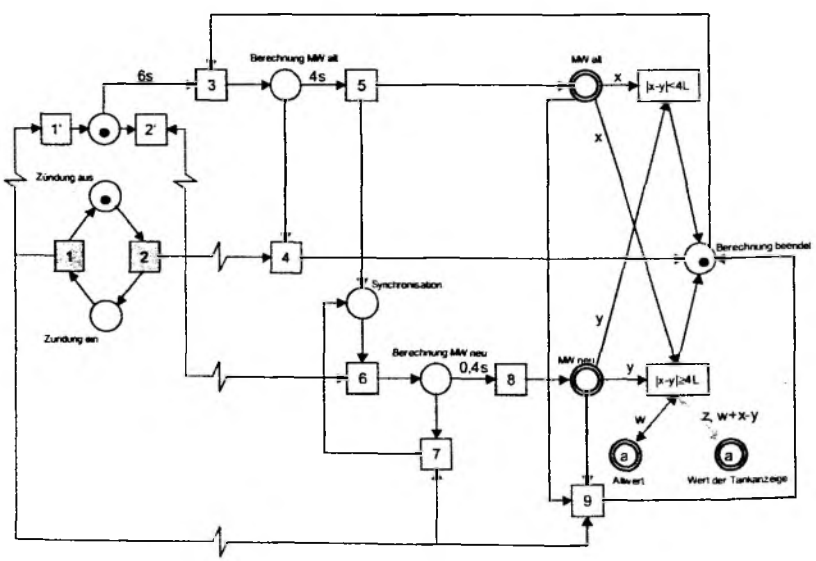

Abbildung 7: Modell der Nachtankerkennung bei *Zündung aus*.

Im linken Modellteil sind die physikalischen und internen Zustände und Zustandsübergänge der Zündung modelliert. Um Messphase 1 zu starten (Transition 3), muss die Zündung für 6 Sekunden

aus sein. Nach 4 Sekunden ist die erste Messphase beendet (Transition 5 schaltet), welche den Mittelwert *MWalt* produziert.

Beim Einschalten der Zündung wird Messphase 2 gestartet (Transition 6), falls Messphase 1 schon beendet ist (Stelle *Synchronisation* ist markiert). Falls die Zündung während der Berechnungsdauer (0, 4 Sekunden) eingeschaltet bleibt, wird *MWneu* berechnet.

Bleibt die Zündung eingeschaltet, wird, falls die Differenz von MWalt und MWneu wenigstens 4 Liter ist, auf Nachtanken erkannt (die Transition mit der Schaltbedingung $|x-y| \geqslant 4L$ schaltet, die Differenz wird zum bisherigen Tankinhalt in der Stelle *Wert der Tankanzeige* addiert), andernfalls wird kein Nachtanken erkannt (die Transition mit der Schaltbedingung $|x-y| < 4L$ schaltet, der Tankinhalt bleibt unverändert).

Gleichzeitig wird die Stelle *Berechnung beendet* (deren Marke beim Schalten der Transition 3 gelöscht wurde) wieder markiert, so dass beim nächsten Ausschalten der Zündung eine neue Nachtankerkennung starten kann.

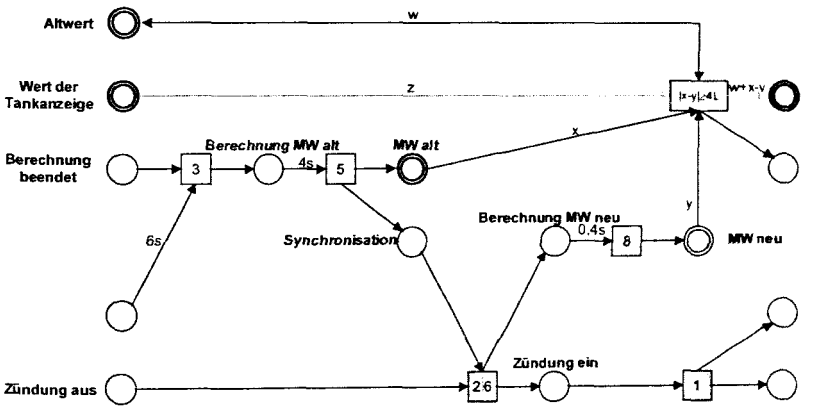

Abbildung 8: Durchführung der Nachtankerkennung, bei der auf Nachtanken erkannt wird.

Anforderung (ii) soll als Beispiel für die Formalisierung und Verifikation einer Anforderung dienen. Sie lautet formal entsprechend dem vorgestellten Modell:

Wenn genau die Stellen MWalt, Zündung aus, Wert der Tankanzeige, Altwert und Synchronisation markiert sind und dann die Transitionen 2 und 1 innerhalb von 0, 4 Sekunden aufeinander schalten, so bleibt die Markierung in der Stelle MWalt gleich

Eine hinreichende Vorbedingung ist der Umstand, dass von den drei Stellen *Berechnung MWalt*, *MWalt* und *Berechnung beendet* stets genau eine markiert ist. Da in der zu verifizierenden Anforderung die Stelle *MWalt* markiert ist, tragen die beiden anderen Stellen keine Marken. Somit können die Transition 3 und 4 nicht schalten, solange *MWalt* markiert bleibt.

Die einzige aktivierte, nicht synchronisierte Transition ist damit Transition 2, welche aufgrund der Progressannahme schaltet.

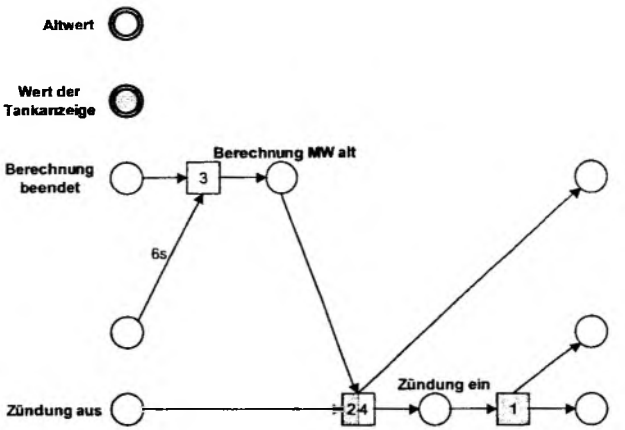

Abbildung 9: Abbruch der Nachtankerkennung während der ersten Messphase durch Einschalten der Zündung.

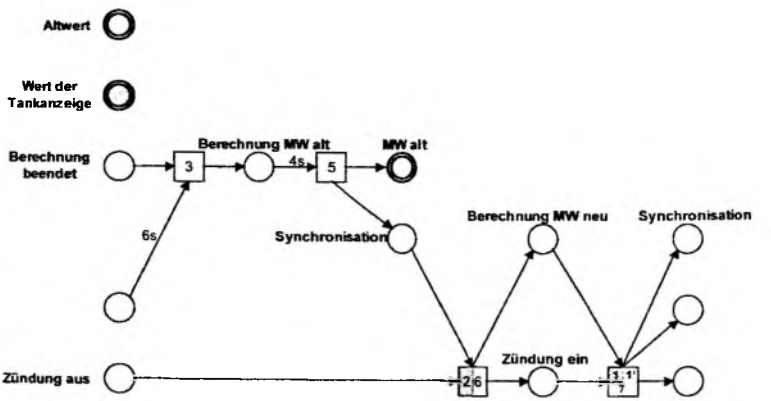

Abbildung 10: Unterbrechung der Nachtankerkennung während der zweiten Messphase durch kurzfristiges Einschalten der Zündung.

Da Transition 6 auch aktiviert ist, wird sie von Transition 2 synchronisiert. Dadurch wird die Stelle *Berechnung MWneu* markiert. Per Voraussetzung schaltet Transition 1 innerhalb von 0,4 Sekunden nach Transition 2. Da die Marke auf der Stelle *Berechnung MWneu* für 0,4 Sekunden erhalten bleibt, bevor Transition 8 schalten kann, ist Transition 7 zum Zeitpunkt des Schaltens von Transition 1 aktiviert, und wird somit von Transition 1 synchronisiert. Dadurch wird die Marke in *Berechnung MWneu* gelöscht und eine Marke in der Stelle *Synchronisation* produziert, was die Anfangsmarkierung wiederherstellt. Diese Schaltfolge kann beliebig oft wiederholt werden, der Wert der Stelle *MWalt* bleibt erhalten.

6 Modell der Nachtankerkennung bei *Zündung ein* und *Fahrzeug steht*

Die Nachtankerkennung bei *Zündung ein* und *Fahrzeug steht* erfolgt ebenfalls in zwei Messphasen. Messphase 1 wird eingeleitet, wenn bei eingeschalteter Zündung 8 Sekunden lang Fahrzeugstillstand vorliegt. Diese Phase dauert 4 Sekunden und liefert den Mittelwert *MWalt*. Messphase 2 liefert im Anschluss laufend einen neuen Mittelwert *MWneu*.
Wenn einmal Nachtanken erkannt wurde (die Differenz der beiden Mittelwerte größer als 4 Liter ist), wird der Tankinhalt laufend neu aus den *MWneu*-Werten berechnet, indem die Differenz der Mittelwerte *MWneu* und *MWalt* zum Altwert addiert wird.
Folgende Anforderungen (u.a.) sind zu erfüllen:

(i) Wenn einmal Nachtanken erkannt wurde, verkürzt sich die Zeitspanne zur Ermittlung der Mittelwerte *MWNeu* von 2 Sekunden auf 0,4 Sekunden.

(ii) Die Berechnung der Mittelwerte *MWneu* dauert an, bis das Fahrzeug wieder fährt oder die *Zündung ein*geschaltet wird.

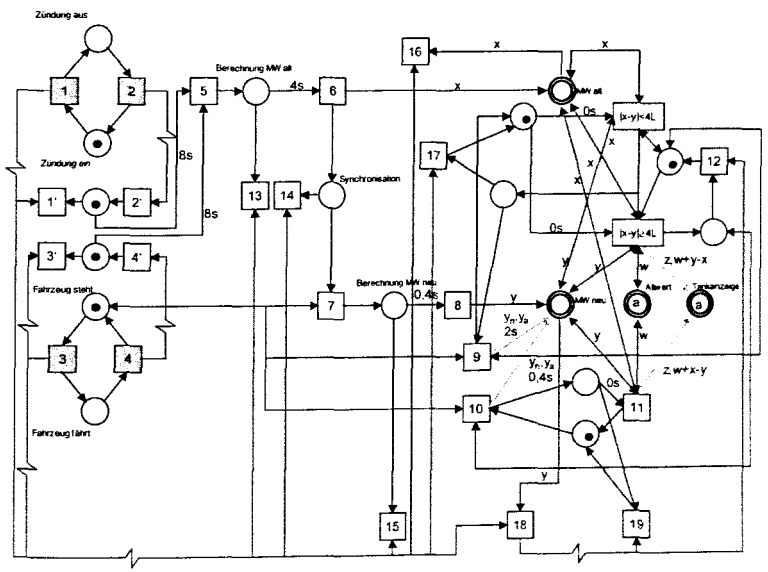

Abbildung 11: Modell der Nachtankerkennung bei *Zündung ein* und *Fahrzeug steht*.

Im linken Modellteil sind die physikalischen und internen Zustände und mögliche Zustandsübergänge der Zündung und des Fahrzeugs (Fahrzeug steht, Fahrzeug fährt) dargestellt. Um Messphase 1 zu starten (Transition 5), muss 8 Sekunden lang die Zündung aus sein und das Fahrzeug stehen. Die Berechnung von *MWalt* ist nach 4 Sekunden beendet (Transition 6 schaltet). Danach

beginnt die Berechnung des Mittelwerts *MWneu* (Transition 7). Diese ist nach 0,4 Sekunden beendet (Transition 8), falls die Zündung solange eingeschaltet bleibt.

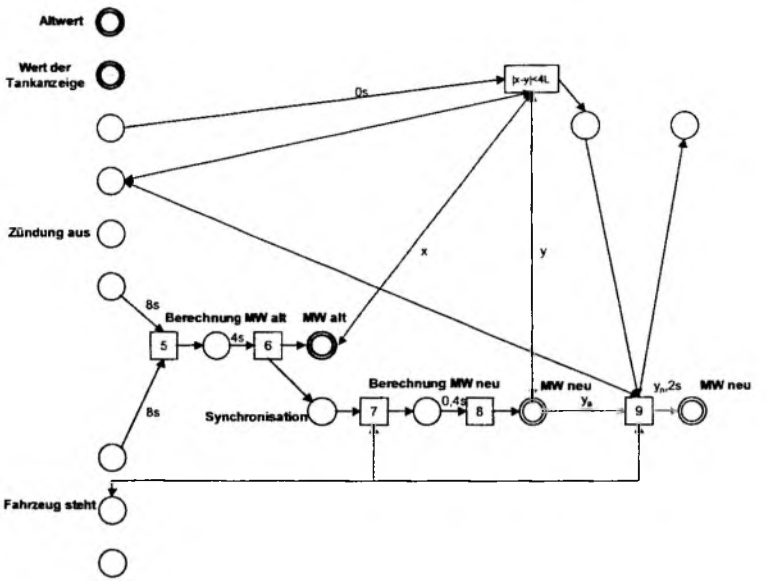

Abbildung 12: Vollständige Durchführung der Nachtankerkennung ohne Erkennung auf Nachtanken.

Unmittelbar nach der Berechnung von *MWneu* (Zeitbeschriftung 0 *Sekunden*), wird die Differenz von *MWalt* und *MWneu* daraufhin geprüft, ob sie $\geqslant 4$ Liter oder < 4 Liter ist, und entsprechend wird auf Nachtanken (Transition mit der Schaltbedingung $|x - y| \geqslant 4L$ schaltet) bzw. auf kein Nachtanken (Transition mit der Schaltbedingung $|x - y| < 4L$ schaltet) erkannt. Wird auf kein Nachtanken erkannt, wird alle 2 Sekunden ein neuer Mittelwert *MWneu* berechnet (Transition 9) und obiger Ablauf wiederholt.
Sobald einmal auf Nachtanken erkannt wurde, wird alle 0,4 Sekunden ein neuer Mittelwert *MWneu* berechnet (Transition 10) und jeweils unmittelbar (Zeitbeschriftung 0 Sekunden) der Wert der Tankanzeige durch Addition der Differenz der Mittelwerte zum Altwert aktualisiert (Transition 11), bis entweder die Zündung ausgeschaltet wird (Transition 1) oder das Fahrzeug anfährt (Transition 3). In diesen Fällen wird die Neuberechnung des Werts der Tankanzeige gestoppt und die Ausgangsmarkierung im rechten Modellteil wiederhergestellt, so dass beim nächsten Fahrzeugstillstand bei *Zündung ein* nach 8 Sekunden eine neue Nachtankerkennung beginnen kann.
Als Beispiel sei Anforderung (i) formalisiert und ihre Verifikation skizziert:

Nach dem Schalten der Transition mit der Schaltbedingung $|x - y| \geqslant 4L$ *darf Transition* 9 *nicht mehr, dafür aber Transition* 10 *schalten. Diese schaltet im Wechsel mit Transition* 11 *bis irgend-*

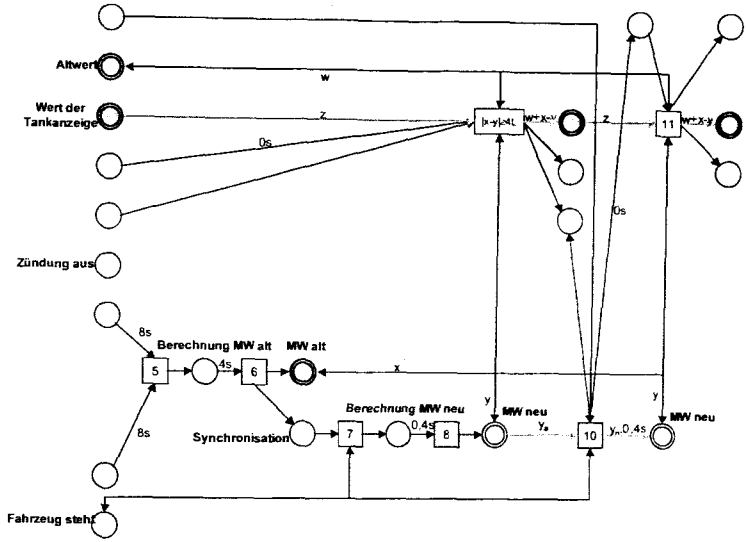

Abbildung 13: Vollständige Durchführung der Nachtankerkennung mit Erkennung auf Nachtanken.

wann Transition 1 *oder Transition* 3 *schaltet*

Zu Beginn der Nachtankerkennung ist die Stelle zwischen Transition 12 und der Transition mit der Schaltbedingung $|x - y| \geqslant 4L$ markiert. Diese Markierung ist Voraussetzung für das Schalten von Transition 9 (Ermittlung der *MWneu*-Werte im Abstand von 2 Sekunden). Analog dazu ist die Markierung der Stelle zwischen der Transition mit der Schaltbedingung $|x - y| \geqslant 4L$ und Transition 12 Voraussetzung für das Schalten von Transition 10. Da immer genau eine der beiden Stellen mit genau einer Marke markiert ist (diese Eigenschaft ist durch sogenannte Stelleninvarianten formal nachweisbar), gilt: Falls Transition 9 schalten kann, so kann Transition 10 nicht schalten, und umgekehrt. Nach dem Schalten der Transition mit der Schaltbedingung $|x - y| \geqslant 4L$ kann also nur noch Transition 10 im Wechsel mit Transition 11 schalten, solange bis Transition 12 schaltet. Diese wird von den Transitionen 1 und 3 synchronisiert.

7 Modell der Geberfehler-Behandlung

Ein Geber kann sich in drei verschiedenen Zuständen befinden: Geber intakt, Geberbruch oder Kurzschluss. Desweiteren sind für den ADC-Wert (das ist der Geberwert) die Werte *untere Schwelle* (Su), *unterer Anschlag* (Au), *oberer Anschlag* (Ao) und *obere Schwelle* (So) definiert. Liegt der ADC-Wert ausserhalb der Schwellwerte, so soll auf Geberfehler erkannt werden (auf Geberbruch, falls der ADC-Wert zu groß ist, und auf Kurzschluss, falls der ADC-Wert zu klein ist). Liegt der

ADC-Wert zwischen Schwellwert und Anschlagwert, so soll für die Tankinhaltsberechnung der Anschlagwert verwendet werden.

Das Modell soll folgende Anforderungen erfüllen:

(i) Es soll eine Geberfehlerbehandlung nur bei *Zündung ein* stattfinden.

(ii) Wird die Zündung eingeschaltet, soll zunächst kein Geberfehler angenommen werden.

(iii) Es soll auf Geberbruch erkannt werden, falls 20 Sekunden lang der ADC-Wert zu groß ($> So$) ist.

(iv) Es soll auf Kurzschluss erkannt werden, falls 20 Sekunden lang ein zu kleiner ($< Su$) ADC-Wert vorliegt.

(v) Es soll von einem Geberfehler auf intakten Geber gewechselt werden, falls 4 Sekunden lang die ADC-Werte wieder innerhalb der Schwellwerte sind.

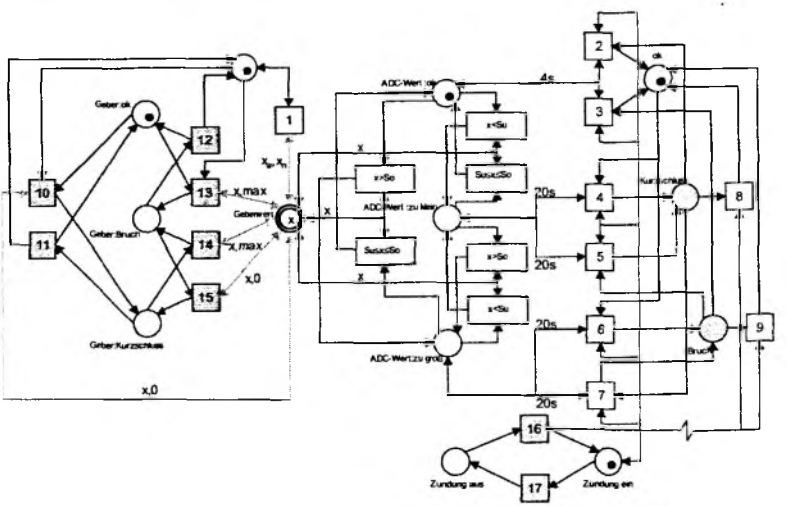

Abbildung 14: Modell der Geberfehlerbehandlung für einen Geber.

Das Modellnetz besteht aus vier Teilen. Im linken Teil sind die möglichen physikalischen Zustände (Geber:ok, Geber:Bruch, Geber:Kurzschluss) und Zustandsübergänge des Gebers modelliert. Nur im Fall, dass der Geber intakt ist, wird der tatsächliche Sensorwert x (wird von Transition 1 geliefert) als ADC-Wert verwendet, ansonsten ein maximaler Wert $\max > So$ (bei Geberbruch) oder ein minimaler Wert $0 < Su$ (bei Kurzschluss). Transition 1 ließe sich noch durch ein Teilnetz verfeinern, um die PT1-Dämpfung des ADC-Wertes und die Verwendung des Anschlagwertes, falls sich der ADC-Wert zwischen Schwellwert und Anschlagwert befindet, sichtbar zu machen.

Unten rechts sind die Zustandsübergänge der Zündung modelliert.

Der mittlere und rechte Teil modellieren den Algorithmus zur Geberfehlerbehandlung. Im mittleren Teil erfolgt die ADC-Wert-Behandlung, der entweder normal (Stelle *ADC-Wert:ok*), zu klein

(Stelle *ADC-Wert:zu klein*) oder zu groß (Stelle *ADC-Wert:zu groß*) ist, in Abhängigkeit des gemessenen Geberwertes in der Stelle *Geberwert*. Hierzu sind die Transitionen mit den entsprechenden Schaltbedingungen versehen.

Im rechten Teil erfolgt (nach gewissen Pufferzeiten) abhängig von dieser Einteilung die interne Erkennung auf einen Geberfehler und die Art des Geberfehlers bzw. auf Intaktheit des Gebers. Letztere Erkennung wird nur durchgeführt, wenn die Zündung eingeschaltet ist (Stelle *Zündung ein*).

Exemplarisch werden in den folgenden Abbildungen durch halbgeordnete Abläufe gegebene Ablaufszenarien obigen Modells vorgestellt.

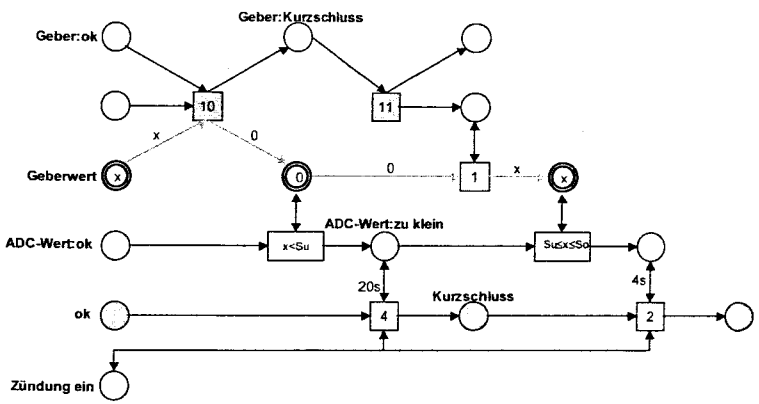

Abbildung 15: Sequentielle Erkennung auf Kurzschluss und Intaktheit des Gebers, ausgelöst durch die entsprechenden physikalischen Zustandsübergänge des Gebers.

Exemplarisch sei Anforderung (iii) formalisiert und verifiziert:

Wenn für mindestens 20 Sekunden die Stellen Geber:Bruch und Zündung ein gleichzeitig markiert sind, so wird nach diesen 20 Sekunden die Stelle Bruch markiert sein

Offenbar muss Eigenschaft (i) als eine Grundvoraussetzung für eine Geberfehlerbehandlung integriert werden. Die Aussage wurde ausserdem dahingehend verallgemeinert, dass der Modellteil für die physikalischen Zustandsübergänge des Gebers mit berücksichtigt wurde (was in der informalen Formulierung der Anforderung (iii) nicht der Fall war).

Folgende Vorbetrachtungen benötigt man für die Verifikation dieser Aussage:

Man stellt fest, dass im mittleren und rechten Modellteil zu jedem Zeitpunkt jeweils genau eine Stelle mit genau einer Marke versehen ist (die jeweiligen Stellenmengen bilden also Stelleninvarianten). Weiterhin ist in diesen beiden Modellteilen zu jedem Zeitpunkt jeweils genau eine Transition aktiviert, was insbesondere bedeutet, dass es keine Konflikte zwischen diesen Transitionen geben kann.

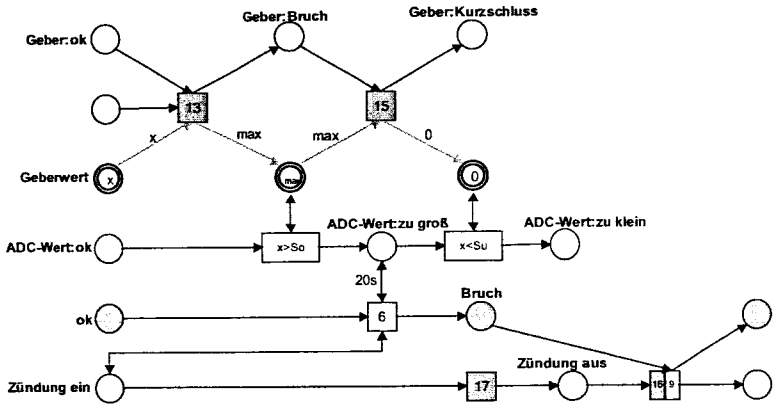

Abbildung 16: Erkennung auf Bruch des Gebers, ausgelöst durch den physikalischen Zustandsübergang des Gebers. Ein anschließender Kurzschluss wird nicht mehr erkannt, da die Zündung aus ist.

Um die Stelle *Bruch* zu markieren, muss eine der beiden Transitionen 13 oder 14 geschaltet haben. Insbesondere befindet sich dann in der Stelle *Geberwert* der Wert max. Da die Stelle *Bruch* für 20 Sekunden markiert bleibt, schaltet für diese Zeit keine weitere Transition aus dem linken Modellteil. Somit befindet sich auch die Stelle *Geberwert* für (mindestens) 20 Sekunden der Wert max. Aufgrund der Vorbetrachtungen kann im mittleren Modellteil nun genau eine der Transitionen mit der Bedingung $x > So$ schalten. Sie schaltet aufgrund der Progressannahme, da sie mit keiner Transition aus anderen Modellteilen in Konflikt stehen (zwischen den Modellteilen gibt es keine Flusskanten). Danach ist die Stelle *ADC-Wert:zu groß* markiert und bleibt dies aufgrund obiger Betrachtungen für (mindestens) 20 Sekunden (für diese Zeit ist keine weitere Transition aktiviert, die die Markierung verändern könnte). Entweder ist jetzt im rechten Modellteil die Stelle *Bruch* schon markiert und somit keine Transition im rechten Modellteil aktiviert, oder es ist im rechten Modellteil aufgrund der Vorbetrachtungen genau eine der Transitionen 6 oder 7 aktiviert (je nachdem ob vorher der ADC-Wert *ok* oder zu klein war), welche dann wieder aufgrund der Progressannahme nach 20 Sekunden schaltet und die Stelle *Bruch* markiert.

Für das Modell der Geberfehler-Behandlung soll exemplarisch auf die Generierung von Testfällen bzw. Testvektoren eingegangen werden:

Die Werte für die Eingabekomponenten eines Testvektors werden von den grau bzw. hellgrau unterlegten Transitionen geliefert und entsprechen der Markierung gewisser Stellen. Diese Stellen repräsentieren Sensordaten bzw. Schnittstellen zu entsprechenden Komponenten, die Sensordaten liefern oder Kontrollparameter. In diesem Modell ist (Geber:ok, Zündung ein, max) ein Eingabevektor. Hierbei repräsentiert die erste Komponente den Zustand des Gebers (wird von den Transitionen $10 - 15$ geliefert), die zweite Komponente den Zustand der Zündung (wird von den

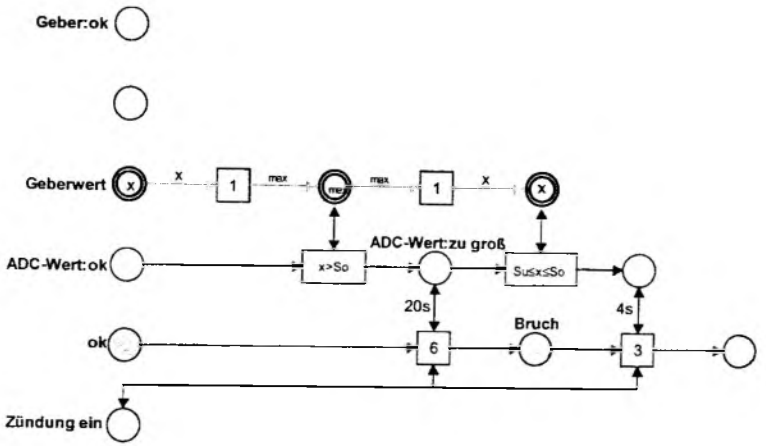

Abbildung 17: Sequentielle Erkennung auf Bruch und Intaktheit des Gebers, ausgelöst durch Messdaten.

Transitionen 16, 17 geliefert) und die dritte Komponente den Geberwert (wird von Transition 1 geliefert).

Die Werte der Ausgabekomponenten entsprechen der Markierung der grau unterlegten Stellen. Geeignete Werte für die Testvektoren bzw. für Folgen von Testvektoren lassen sich aus im Modell dargestellten Wertebereichen und Grenzverhältnissen ablesen. Gegebenenfalls sind dafür zusätzliche Informationen einzuholen, wenn nämlich diese Wertebereiche für den Algorithmus ohne Belang sind (Bsp.: der maximale durch einen Geber anzeigbare Füllstand des Tankes; er wird durch die Dimensionierung des Tankes und der Geberskala bestimmt, ist für die Plausibilitätsüberprüfung wichtig, für andere Algorithmen aber unerheblich).

Für die lokale Testvektorenbestimmung ist aus einem Modell nicht mehr zu gewinnen als explizit hineingeschrieben wurde, was auch nicht überrascht. Die Vorteile der modellbasierten Testvektorenbestimmung liegen vielmehr in globalen Aspekten. Die Berücksichtigung von Abhängigkeit bzw. von Unabhängigkeit von Ein- und Ausgabewerten, die bei der Analyse des Modells gewonnen werden kann, reduziert die Anzahl notwendiger Testwertkombinationen erheblich. Man sieht nun z.B. am Modell, dass zur Erkennung eines Geberfehlers *Zündung ein* und ein *fehlerhafter Geberwert* notwendig sind. Deshalb müssen für den Test nicht alle Eingabekombinationen von (Geber:ok, Geber:Bruch, Geber:Kurzschluss) und (Zündung: ein, Zündung: aus) getestet werden, denn bei *Zündung aus* spielt der erste Eingabeparameter keine Rolle mehr.

Dieser Ansatz basiert auf der Annahme, dass der tatsächlich realisierte Steuerungsalgorithmus dieselben Abhängigkeiten besitzt wie der modellierte Algorithmus. Beide sollen bezüglich der Spezifikation äquivalent sein, diese Äquivalenz ist aber auch nur durch Testreihen zu belegen. Insofern kann die modellbasierte Testvektorengenerierung bestenfalls Hinweise auf relevante Testdaten lie-

fern. Es kann aber nicht ausgeschlossen werden, dass Fehler der Steuerung verborgen bleiben, die bei anderen Testdaten deutlich geworden wären. Nur weitere Informationen zu den realisierten Algorithmen, die nicht der Spezifikation entnommen werden können, lassen auf diese Testdaten schliessen.

8 Werkzeugunterstützung

Zur Unterstützung aller im Vorgehensmodell (Abschnitt 3) genannten Arbeitsschritte sind Werkzeuge notwendig, wenn dieser Ansatz zuverlässig auch für größere Komponenten realisiert werden soll. Ohne Werkzeugeinsatz können die Schritte zwar exemplarisch vollzogen werden, der notwendige Aufwand und die Fehleranfälligkeit würden das Konzept aber in Frage stellen. Die genaue Spezifikation der Werkzeuge hängt ab von der Präzisierung des Konzepts in einigen Schritten (z.B. Testfallgenerierung) einerseits, und andererseits von gewünschten und notwendigen Benutzerschnittstellen. Wie stets bei Werkzeugplanung und -entwicklung ist also zunächst die Frage zu beantworten, welche Personen mit welchem Hintergrund in welcher Situation die Werkzeuge einsetzen sollen.

Der konkrete Stand der Werkzeugentwicklung im Projekt SPECIMEN sieht zur Zeit wie folgt aus [DeJuLo03]: Es existiert ein Werkzeug (VIPtool [Fre01]), das bei Eingabe eines Petri-netz-Modells und von Anforderungen in einer visuellen (Petrinetz-angelehnten) Eingabesprache kausale Abläufe erzeugt, geeignet visualisiert und bezüglich der Anforderungen analysiert. Insbesondere ist dieses Werkzeug in der Lage, das Modell und die Anforderungen iterativ zu validieren. Darüber hinaus existiert eine Erweiterung um Zeit- und Kostenaspekte, die zur Ermittlung von Performance-Aussagen eines Modells (Durchlaufzeit, durchschnittliche Kosten usw.) eingesetzt wird. Das Werkzeug ist plattformunabhängig in der Sprache Python implementiert. Aktuell wird an einer Neuentwicklung in der Sprache Java gearbeitet. In dieser Neuentwicklung werden die auch im Projekt notwendigen Erweiterungen um Signale (Signal-Netze) und das Modularitätskonzept unterstützt.

Langfristig kann das genannte Werkzeug den Kern eines Gesamtsystems für das Projekt darstellen. Die folgenden Ergänzungen sind notwendig:

- Erstellung eines Modells durch Eingabe und Faltung von Abläufen. Dies ist zwar technisch aufwändig, es gibt aber keine grundsätzlichen Probleme bei der Realisierung, mit Ausnahme der graphischen Darstellung des generierten Modells. Aktuelle Graph-Drawing-Algorithmen sind zwar in der Lage, azyklische Netze (z.B. Abläufe) recht lesbar darzustellen, aber nicht beliebige Graphen, wie sie als Gesamtmodelle auftreten. Hier wird also eine Nachbearbeitung per Hand notwendig, die wiederum geeignet durch ein Werkzeug unterstützt werden muss.

- Verifikationskomponenten sollten nicht neu implementiert werden, sondern können durch Einbindung existierender Werkzeuge realisiert werden. Allerdings ist dazu notwendig, dass die Anforderungen in einer temporalen Logik angegeben werden. Dies ist zwar grundsätzlich leicht

möglich (die bisher betrachteten Anforderungen lassen sich in der Linear-Time Logik LTL ausdrücken), aber aufgrund der gewöhnungsbedürftigen Syntax von Logik sollte eine andere Eingabeform und ein Transformationsverfahren entwickelt werden.

- Die Testfallgenerierung benötigt effiziente Algorithmen zur Petrinetz-Analyse. An derartigen Algorithmen und ihrer Implementierung wird zur Zeit am Lehrstuhl für Angewandte Informatik der Katholischen Universität Eichstätt-Ingolstadt gearbeitet.

- Der Einsatz eines Modells als Testreferenz benötigt eine Simulatorkomponente und geeignete Schnittstellen zur Testumgebung. Hier ist der Einsatz eines der vielen existierenden Petrinetz-Simulatoren nicht ohne Weiteres möglich, denn die Semantik der hier eingesetzten Netzklasse unterscheidet sich in vielen Details von anderen verwendeten Netzklassen. Die Realisierung eines Simulators auf Grundlage des existierenden Werkzeuges VIPtool ist aber mit recht geringem Aufwand möglich. Auch die Realisierung geeigneter Schnittstellen zur Testumgebung sollte keine größeren technischen Probleme aufwerfen. Als problematisch könnte sich allerdings der Einsatz in Realzeit erweisen, insbesondere bei sehr zeitkritischen Fahrzeugkomponenten. Da aber bislang weder von der Fahrzeug- noch von der Werkzeugseite laufzeitbezogene Daten vorliegen, sind konkrete Aussagen dazu noch nicht möglich.

- Zu den Benutzungsschnittstellen wurde bereits weiter oben erwähnt, dass diese von den konkreten Benutzern abhängen. Alternative Darstellungsformen, die Integration geeigneter Icons u.s.w. sind leicht möglich. Eine signifikante Vereinfachung der Formalismen führt dagegen notwendigerweise zum Verlust von Aussagekraft der Modelle und Effektivität des Konzeptes. Positiv ausgedrückt: Es sollte recht früh ein Trade-Off zwischen einfachen, intuitiven, adäquaten Benutzungsschnittstellen und angestrebtem Wert der Ergebnisse zwischen beiden Projektpartnern verhandelt werden.

9 Erfahrungen aus der Fallstudie

Grundsätzlich erfuhr der gewählte Ansatz eine positive Rückmeldung durch die Audi AG. Zum einen konnten die meisten gestellten Probleme gelöst werden, zum anderen konnten in den bereitgestellten Unterlagen zahlreiche Unstimmigkeiten und Fehler erkannt werden.
Die Annahme, dass ausgehend von einer vagen Beschreibung der Strecke und einiger Anforderungen ein Steuerungsalgorithmus entwickelt werden soll, hat sich in dieser Fallstudie nur teilweise bestätigt. Im vorliegenden Fall standen sehr genaue Kenntnisse über Teilbereiche der Strecke zur Verfügung. Diese Informationen mussten in die verwendete Modellierungssprache übersetzt werden, was aufgrund impliziter Annahmen zu Problemen führte und zahlreiche Rücksprachen erforderlich machte. Eigenschaften wie Sicherheit oder Lebendigkeit waren in den halbformalen Anforderungen vorwiegend implizit vorhanden oder wurden nicht erwähnt (beispielsweise soll der Fahrer gewarnt werden, bevor der Tank leer ist). Hauptsächlich wurden die Modelle auf der Grund-

lage gewünschte Szenarien der Art "was passiert, wenn..." erstellt. Aus Modellierungssicht kann man sagen, dass gegebene halbformale Abläufe im Rahmen der Fallstudie formalisiert und durch Experten validiert wurden. Jeder dieser Abläufe hat Schnittstellen zum Modell der Strecke. Ein Modell des Steuerungsalgorithmus, das diese Abläufe erlaubt, enthält im allgemeinen auch weitere Abläufe. Es zeigt sich also, dass sich Situationen ergeben können, die nicht in den Szenarien geschildert wurden. Unser Simulationsansatz identifiziert diese neuen Szenarien und zeigt dem Benutzer weitere mögliche Abläufe bezüglich dieses Modells, so dass er angehalten wird, die notwendigen Anforderungen zu vervollständigen. Danach werden die formalisierten Anforderungen validiert.

In diesem Anwendungsbereich spielten Echtzeitaspekte eine wichtige Rolle. Zum Beispiel ist das Verhalten des Tankalgorithmus von der Dauer des Fahrzeugstillstands abhängig. Diese Echtzeitaspekte waren bislang nicht in der verwendeten Modellierungssprache integriert und konnten daher nur indirekt behandelt werden. Momentan arbeiten wir an der Integration dieses Aspekts, wobei die nebenläufige kausale Semantik (vgl. [DeJuLo00]) erhalten bleiben soll. Wir führen für einzelne Komponenten den Begriff der "lokalen Zeit" ein; es wird jedoch keine globale Zeitskala vorgegeben, die eine vollständige Ordnung aller Elemente erzwingen würde.

Die Kooperation mit der Audi AG wird fortgesetzt und weitere steuerungsrelevante Komponenten werden modelliert. Anhand dieser größeren Modelle soll dann Nebenläufigkeit und Kausalität zwischen Ereignissen, die zu unterschiedlichen Komponenten gehören, stärker ausgenutzt werden.

Literatur

[De00] Desel, J.: Validation of Process Models by Construction of Process Nets. In Wil van der Aalst, Jörg Desel, Andreas Oberweis (Eds.): *Business Process Management*, LNCS 1806. Springer-Verlag, S. 110–128, 2000.

[De02] Desel, J.: Model Validation - A Theoretical Issue? In Javier Esparza, Charles Lakos (Eds.): *Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets 2002*, ICATPN 2002, LNCS 2360. Springer-Verlag, S. 23–43, 2002.

[DeJuLo00] Desel J., Juhás G., Lorenz R.: Process Semantics and Process Equivalence of NCEM. In Proc 7. Workshop Algorithmen und Werkzeuge für Petrinetze AWPN 2000, Fachberichte Informatik, Universität Koblenz - Landau, S. 7–12, 2000.

[DeJuLo02] Desel J., Juhás G., Lorenz R.: Input/Output Equivalence of Petri Modules. In Proc. of the 6th Biennial World Conference on Integrated Design and Process Technology IDPT 2002, Pasadena, California, 2002.

[DeJuLo03] Desel J., Juhás G., Lorenz R., Neumair Ch.: Modelling and Validation with VipTool Erscheint in Proc. of the International Conference on Business Process Management, LNCS, Springer-Verlag, 2003.

[DeRe98] Desel J., Reisig W.: Place/Transition Petri Nets. In W. Reisig, G. Rozenberg (Eds.): *Lectures on Petri Nets I: Basic Models*, LNCS 1491. Springer-Verlag, S. 122–173, 1998.

[Fre01] Freytag T.: Softwarevalidierung durch Auswertung von Petrinetz-Abläufen. Dissertation, Universität Karlsruhe, 2001.

[HaLu99] Hanisch H.-M., Lüder A.: Modular Modeling of Closed-Loop Systems. In Proc. of Colloquium on Petri Net Technologies for Modelling Communication Based Systems, S. 103–126, 1999.

[HaLu00] Hanisch H.-M., Lüder A.: A Signal Extension for Petri nets and its Use in Controller Design. *Fundamenta Informaticae*, 41(4), S. 415–431, 2000.

[HaLuRa97] Hanisch H.-M., Lüder A., Rausch M.: Controller Synthesis for Net Condition/Event Systems with a Solution for Incomplete State Obersvation. *European Journal of Control*, (3), S. 280–291, 1997.

[HaThLu97] Hanisch H.-M., Thieme J., Lüder A.: Towards a Synthesis Method for Distributed Safety controllers Based on Net Condition/Event Systems. *Journal of Intelligent Manufacturing*, (5), S. 357–368, 1997.

[HoKrGi97] Holloway L. E., Krogh B. H., Giua A.: A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, (7), S. 151–190, 1997.

[JuLo02] Juhás G., Lorenz R.: Modelling with Petri Modules. In B. Caillaud, Ph. Darondeau, L. Lavagno, X. Xie (Eds.) *Synthesis and Control of Discrete Event Systems*, S. 125–138, Kluwer, 2002.

[Schn99] Schnieder E.: *Methoden der Automatisierung. Beschreibungsmittel, Modellkonzepte und Werkzeuge für Atomatisierungssysteme.* Vieweg, 1999.

[SrKr91] Sreenivas R. S., Krogh B. H.: Petri Net Based Models for Condition/Event Systems. In *Proceedings of 1991 American Control Conference*, vol. 3, S. 2899–2904, Boston, MA, 1991.

[St00] Starke P. H.: Das Komponieren von Signal-Netz Systemen. In Proc 7. Workshop Algorithmen und Werkzeuge für Petrinetze AWPN 2000, Fachberichte Informatik, Universität Koblenz - Landau, S. 1–6, Oktober 2000.

[ZhDiC93] Zhou M. C., DiCesare F.: *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer, 1993.