



HELMUT SCHMIDT
UNIVERSITÄT

Universität der Bundeswehr Hamburg

Automatische Generierung von IEC 61131-3 Steuerungscode aus einer GRAFCET-Spezifikation

Von der Fakultät für Maschinenbau
der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg
zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte

DISSERTATION
vorgelegt von

Dipl.-Ing. Frank Schumacher

aus Siegen

Hamburg, 2013

Gutachter: Univ.-Prof. Dr.-Ing. Alexander Fay
Univ.-Prof. Dr.-Ing. Georg Frey

Tag der mündlichen Prüfung: 27. September 2013

Vorwort des Autors

Die vorliegende Arbeit entstand während meiner Zeit als Wissenschaftlicher Mitarbeiter an der Professur für Automatisierungstechnik der Helmut-Schmidt-Universität/ Universität der Bundeswehr Hamburg von Januar 2011 bis Dezember 2013.

Ich möchte nachfolgend all den Menschen meinen Dank aussprechen, die maßgeblich zum Entstehen dieser Arbeit beigetragen haben.

Mein besonderer Dank gilt Herrn Professor Dr.-Ing. Alexander Fay. Als Doktorvater hat er mich stets unterstützt und gefördert und mir in zahlreichen Gesprächen und Diskussionen vielfältige Anregungen und Impulse gegeben. Durch ihn wurde diese Arbeit erst möglich.

Herrn Professor Dr.-Ing. Georg Frey danke ich für das Interesse an meiner Arbeit und die Übernahme des Zweitgutachtens. Herrn Professor Dr.-Ing. Klaus Krüger danke ich für die Übernahme des Prüfungsvorsitzes.

Weiterhin gilt meinen Mitarbeiterkolleginnen und -kollegen großer Dank für die gemeinsame Zeit am Institut, die durch eine stets offene und respektvolle Zusammenarbeit und Arbeitsatmosphäre geprägt war. Hervorheben möchte ich an dieser Stelle Karin Eckert, Sebastian Schreiber und Sebastian Schröck, die mir beim Schreiben der Arbeit wertvolle Anmerkungen, Ratschläge und Korrekturhinweise gaben.

Allen Studenten, die ich als Wissenschaftlicher Mitarbeiter betreuen durfte, möchte ich ebenfalls meinen Dank aussprechen. Ohne ihre Tatkraft wäre der Umfang an Implementierung nicht möglich gewesen.

Nicht zuletzt möchte ich meinen Freunden und meiner Familie für das Verständnis, die Zuversicht und Unterstützung danken, die sie mir in den vergangenen Jahren entgegen gebracht haben. Insbesondere meinen Eltern und meiner Schwester möchte ich dafür danken, dass sie stets an mich geglaubt und mich fortwährend in meinem Tun bestärkt haben, obwohl sie dadurch häufig auf mich verzichten mussten.

Der größte Dank aber gilt meiner Frau Jenny, die mir mit ihrer Liebe, Zuneigung und Geborgenheit die nötige Kraft und den Ansporn gab, alle Entbehrungen und Mühen der letzten Jahre zu meistern.

Hamburg, im Oktober 2013

Frank Schumacher

Inhaltsverzeichnis

Abkürzungsverzeichnis.....	VII
1 Einleitung und Motivation	1
1.1 Motivation	1
1.2 Wissenschaftlicher Beitrag und Gliederung der Arbeit.....	4
2 Formale Methoden im Steuerungsentwurf.....	7
2.1 Steuerungsentwurf in der heutigen Praxis	7
2.2 Steuerungsentwurf auf der Basis grafischer Beschreibungsmittel	8
2.2.1 Grundlagen	8
2.2.2 Verifikation und Test	10
2.2.3 Steuerungssynthese	11
2.2.4 Automatische Generierung.....	12
2.3 Forschungsansätze zur automatischen Generierung.....	13
2.3.1 Domänenunabhängige Beschreibungsmittel	13
2.3.2 Domänenspezifische Beschreibungsmittel.....	15
2.4 Zwischenfazit.....	17
3 Sequential Function Charts gemäß IEC 61131-3.....	19
3.1 Grundlagen zu Sequential Function Charts	19
3.2 Struktur und Wirkungsteil	20
3.3 Dynamisches Verhalten	21
3.4 Möglichkeiten zur hierarchischen Strukturierung	23
3.5 Zeitabhängige Bedingungen	25
3.6 Werkzeugunterstützung	26
4 GRAFCET gemäß IEC 60848.....	28
4.1 Hintergründe	28
4.1.1 Der Weg zur internationalen Norm	28
4.1.2 Anwendungsbereich von GRAFCET	30
4.2 Grundlegende Eigenschaften	31
4.2.1 Struktur.....	31
4.2.2 Wirkungsteil	32
4.2.3 Dynamisches Verhalten.....	35
4.3 Möglichkeiten zur hierarchischen Strukturierung	37
4.3.1 Makroschritte	37
4.3.2 Einschließende Schritte	39
4.3.3 Zwangssteuernde Befehle	40
4.4 Zeitabhängige Bedingungen	42
4.5 Werkzeugunterstützung	44
4.6 Zwischenfazit.....	45

5	Stand der Forschung und Handlungsbedarf bezüglich GRAFCET	47
5.1	Stand der Forschung	47
5.1.1	Ansätze zur Verifikation und Validierung	47
5.1.2	Ansätze zur Steuerungssynthese	50
5.1.3	Ansätze zur automatischen Generierung.....	51
5.2	Handlungsbedarf bezüglich GRAFCET	55
6	Mathematische Basis für das Beschreibungsmittel GRAFCET	60
6.1	Grundlegende formale Definition von GRAFCET als SIPN	60
6.1.1	Das formale Modell eines Basic-Grafcet	60
6.1.2	Berücksichtigung von Makroschritten	68
6.2	Erweiterung des formalen Modells.....	69
6.2.1	Kontinuierlich und gespeichert wirkende Aktionen.....	69
6.2.2	Einschließende Schritte	71
6.2.3	Zwangssteuernde Befehle	74
6.2.4	Zeitabhängige Bedingungen.....	77
6.3	Zwischenfazit.....	82
7	Implementierungsunabhängige Notation für GRAFCET	83
7.1	Voraussetzungen für eine implementierungsunabhängige Notation	83
7.1.1	Anforderungen an eine implementierungsunabhängige Notation.....	83
7.1.2	Vergleich etablierter Auszeichnungssprachen	86
7.2	Konzept für eine implementierungsunabhängige Notation in PNML.....	91
7.3	Umsetzung in die PNML-Notation nach ISO/IEC 15909-2.....	94
8	Transformation von GRAFCET in Steuerungscode gemäß IEC 61131-3	97
8.1	Anforderungen an die Transformation	97
8.2	Konzept für die Transformation	99
8.2.1	Transformation auf Modellebene	99
8.2.2	Transformation auf Metamodellebene	102
8.3	Definition der Transformationsregeln	105
8.3.1	Transformationsregeln auf Modellebene	105
8.3.2	Transformationsregeln auf Metamodellebene.....	107
9	Werkzeugunterstützung	110
9.1	Werkzeug zur Erstellung von GRAFCET-Spezifikationen.....	110
9.1.1	Struktureller Aufbau und Funktionsweise.....	110
9.1.2	Erstellung einer GRAFCET-Spezifikation.....	112
9.2	Werkzeug für die Transformation	115
9.2.1	Struktureller Aufbau und Funktionsweise.....	115
9.2.2	Durchführung der Transformation	119

10 Anwendungsbeispiel	121
10.1 Anlagenbeschreibung und textuelle Spezifikation	121
10.2 Spezifikation des Steuerungsablaufs mit GRAFCET	125
10.3 Transformation in IEC 61131-3 Steuerungscode	127
11 Zusammenfassung und Ausblick.....	131
11.1 Zusammenfassung	131
11.2 Ausblick.....	134
Anhang A Historische Entwicklung der IEC 60848.....	136
Anhang B Aktionsbestimmungszeichen gemäß DIN EN 61131-3.....	137
Anhang C Aktionen und Transitionsbedingungen in GRAFCET	138
Anhang D Transformationsregeln	140
Anhang E Signalübersicht des Prüfautomaten.....	146
Anhang F GRAFCET-Spezifikation des Prüfautomaten	148
Anhang G Struktur des automatisch generierten SFC des Prüfautomaten.....	152
Literaturverzeichnis.....	155
Referenzierte Normen, Richtlinien und Empfehlungen	168
Referenzierte Internetquellen und Software	170
Veröffentlichungen des Verfassers	172
Studentische Arbeiten	172
Lebenslauf.....	174

Abkürzungsverzeichnis

ADEPA	Agence pour le Développement de la Productique Appliquée à l'industrie
AFNOR	Association Francaise de Normalisation
AF CET	Association Francaise pour la Cybernétique Economique et Technique
CPU	Central Processing Unit, zentrale Verarbeitungseinheit eines Mikrocontrollers
CENELEC	Comité Européen de Normalisation Electrotechnique
CAEX	Computer Aided Engineering eXchange
DC / FT	Dependency Charts / Function Table
DIN	Deutsche Industrie Norm
XML	Extensible Markup Language
FBS	Funktionsbausteinsprache gemäß DIN EN 61131-3
GMA	Gesellschaft Mess- und Automatisierungstechnik im VDI/VDE
GRAF CET	GRAPhe Fonctionnel de Commande Etape Transition
Grafcet	Instanz einer mit GRAFCET erstellten Spezifikation
GEMMA	Guide d'Étude des Modes de Marches et d'Arrêts
IEC	International Electrotechnical Commission
ISO	International Organisation for Standardization
MDD	Model Driven Development
MDE	Model Driven Engineering
NCES	Net Condition / Event Systems
AWL	Programmiersprache Anweisungsliste gemäß IEC 61131-3
KOP	Programmiersprache Kontaktplan gemäß IEC 61131-3
POE	Programmorganisationseinheit gemäß IEC 61131-3
SFC	Sequential Function Chart gemäß IEC 61131-3
SPS	Speicherprogrammierbare Steuerung gemäß IEC 61131
SIPN	Steuerungstechnisch Interpretiertes Petrinetz
ST	Strukturierter Text gemäß DIN EN 61131-3
SCT	Supervisory Control Theory
ISA	The International Society of Automation
UML-PA	UML for Process Automation
UML	Unified Modeling Language
VDE	Verband der Elektrotechnik Elektronik Informationstechnik e.V.
VDMA	Verband Deutscher Maschinen- und Anlagenbau e.V.
VDI	Verein Deutscher Ingenieure
VHDL	Very High Speed Integrated Circuit Hardware Description Language
W3C	World Wide Web Consortium

1 Einleitung und Motivation

1.1 Motivation

Seit ihrer Markteinführung zu Beginn der 1970er Jahre haben *SpeicherProgrammierbare Steuerungen* (SPS) Einzug in weite Teile der Automatisierung von Anlagen, wie beispielsweise verfahrenstechnische oder fertigungstechnische Maschinen und Anlagen, gehalten und sind heutzutage in industriellen Steuerungsapplikationen entsprechend weit verbreitet. Aufgrund zunehmender Leistungsfähigkeit und weitestgehend standardisierter Hardware können moderne SPSen als softwareintensive Systeme angesehen werden, deren Steuerungslogik in Form spezieller Programme projektiert und umgesetzt wird [LITZ & FREY 1999]. Ein wesentlicher Anteil der Wertschöpfung im Engineering automatisierter Anlagen ist daher auf das Engineering der Steuerungssoftware zurückzuführen, wie beispielsweise Ergebnisse einer Befragung von Unternehmen des Maschinen- und Anlagenbaus in [VDMA 2012][@] zeigen. Die hinsichtlich des Engineerings notwendigen Arbeitsschritte auf dem Weg von der Anforderungserhebung bis zur Realisierung des lauffähigen Steuerungsprogramms werden unter dem Begriff des **Steuerungsentwurfs** [LITZ & FREY 1999] zusammengefasst, dessen Phasenmodell in Abbildung 1-1 schematisch dargestellt ist.



Abbildung 1-1: Phasen des Steuerungsentwurfs

Der Steuerungsentwurf ist ein wesentlicher Bestandteil des Engineerings automatisierter Anlagen, welches in jüngster Vergangenheit zunehmend den gestiegenen Wettbewerbsanforderungen unterliegt, Lösungen von höherer Qualität in geringerer Zeit und zu geringeren Kosten zu produzieren [HAPPACHER 2012]. Demzufolge ergibt sich, durch eine zunehmende Verflechtung von Automatisierungssystemen und der damit einhergehenden steigenden Komplexität der Automatisierungsaufgaben [AUTOMATION 2012][@], eine steigende Notwendigkeit für geeignete Beschreibungsmittel, Methoden und Werkzeuge [SCHNIEDER 1999], die eine systematische Analyse und Lösung der Automatisierungsaufgabe (**Spezifikation**) sowie eine nachfolgende effiziente Überführung in ausführbare Steuerungsprogramme (**Implementierung**) unterstützen. Grafische Beschreibungsmittel besitzen in diesem Zusammenhang großes Potential für eine gewerkeübergreifende Darstellung relevanter Sachverhalte, insbesondere für die Erstellung von **Modellen** des Steuerungsablaufs, sofern sie einen anwendungsfallbezogenen, eindeutig definierten Zeichenvorrat besitzen. Auch die Art der grafischen Repräsentation trägt entscheidend zur Eingängigkeit des Modells und einem gemeinsamen Verständnis über den beschriebenen Sachverhalt bei [MOODY 2009]. Unterliegen diese Beschreibungsmittel zusätzlich einer mathematisch eindeutigen, formalen Definition, so können die zunächst ausschließlich grafisch transportierten Informationen mit Hilfe eines Datenmodells einer rechnergestützten Handhabung zugänglich gemacht werden.

Für softwareintensive Systeme aus dem Bereich der Informatik sind solche modellgetriebenen Ansätze zur Softwareentwicklung bereits seit längerem etabliert. Sie werden unter dem Begriff des *Model Driven Development* (MDD) zusammengefasst und liefern im Zuge einer

zunehmenden „*Informatisierung*“ der Automatisierungstechnik [JASPERNEITE 2012][@] wesentliche Impulse für einen systematischen Steuerungsentwurf. MDD besitzt die grundlegende Zielsetzung, die mit Hilfe eines grafischen Beschreibungsmittels erstellten Modelle einerseits als Dokumentation des Programms zu pflegen und andererseits die im Modell hinterlegten Informationen rechnergestützt zu interpretieren, zu verarbeiten und in lauffähige Anwendungsprogramme zu transformieren [SELIC 2003]. Allerdings muss bezüglich der Anwendbarkeit von MDD im Engineering automatisierter Anlagen ein wesentlicher Aspekt berücksichtigt werden. So erfüllen Steuerungsprogramme in der Regel eine konkrete technische Aufgabe innerhalb eines technischen Systems, der Anlage. Die in den Steuerungsprogrammen implementierte Logik wirkt sich unmittelbar, beispielsweise in Form von Signalen, auf real existierende elektrische oder mechanische Anlagenkomponenten bzw. Anlagenteile aus und beeinflusst deren Verhalten sowie den in der Anlage ablaufenden Prozess. Im Gegensatz zu einer reinen Softwareentwicklung bestehen diesbezüglich vielschichtige projekt- und anlagenspezifische Abhängigkeiten, die eine isolierte Spezifikation und Implementierung der Steuerungslogik unmöglich machen. Im Rahmen des Steuerungsentwurfs besteht zusätzlich die Notwendigkeit, die an den Schnittstellen betroffener Gewerke und Phasen des Anlagenlebenszyklus auszutauschenden Anlagen- und Prozessinformationen zu berücksichtigen, um bereits vorliegende Engineering-Ergebnisse gewinnbringend, im Sinne eines effizienteren Arbeitsablaufs, zu nutzen. Für eine systematische Verbesserung des Steuerungsentwurfs ergibt sich dann die Möglichkeit der automatischen Generierung von Steuerungscode. Inspiriert durch MDD könnte somit insbesondere die Engineering-Schnittstelle zwischen Spezifikation (**Planung**) und Implementierung (**Realisierung**) in geeigneter Art und Weise verbunden und der Anteil wertschöpfender Tätigkeiten im Steuerungsentwurf durch Reduktion nicht-wertschöpfender Tätigkeiten, nämlich der manuellen Implementierung entsprechend einer vorgegebenen Spezifikation, gesteigert werden. Die für den Steuerungsentwurf wesentlichen Aspekte der Automatisierungslösung, wie beispielsweise der Steuerungsablauf, würden mit Hilfe eines grafischen Modells im Rahmen der Spezifikation möglichst technologieneutral beschrieben und die Implementierung anschließend durch rechnergestützte Methoden (teil-) automatisch generiert. Unter Berücksichtigung dieser Rahmenbedingungen eines systematischen Steuerungsentwurfs kommt den grafischen Beschreibungsmitteln eine Schlüsselrolle für die Etablierung eines *Model Driven Engineerings* (MDE) [FREY & THRAMBOULIDIS 2011] zu.

In der heute üblichen Praxis des Steuerungsentwurfs basiert der Informationsaustausch an der Schnittstelle zwischen Spezifikation und Implementierung allerdings auf Dokumenten, die größtenteils manuell ausgewertet und interpretiert werden müssen und an der Gewerkegrenze zwischen Planung und Realisierung übergeben werden [LOHMANN 2011]. Der zu realisierende Steuerungsablauf wird darin nur ansatzweise grafisch und überwiegend informell durch Anforderungen im Textformat (**informelle Spezifikation**) beschrieben. Modellgetriebene Ansätze hingegen sind kaum relevant und von Seiten der Anwender wenig akzeptiert. So basieren beispielsweise formale Ansätze mit Petrinetzen [WAGNER ET AL. 2007] oder Zustandsautomaten [NKE & LUNZE 2013] auf einer eindeutigen, mathematischen Definition des Beschreibungsmittels (**formales Modell**) und sind dadurch rechnergestützten, formalen Methoden unmittelbar zugänglich. Allerdings können mit solchen, grundsätzlich domänenunabhängigen Beschreibungsmitteln aufgrund ihrer vielseitigen Anwendungsmöglichkeiten und des vergleichsweise hohen Abstraktionsgrades oftmals nur einfache

Modelle erstellt werden, die lediglich einen generellen Blick auf die Automatisierungslösung erlauben. Zudem erfolgen, gemäß den Ausführungen in [VOGEL-HEUSER ET AL. 2013], der Entwurf und die Planung einer neuen Steuerungssoftware meistens durch Ingenieure, die aufgrund ihrer akademischen Ausbildung Kenntnisse über modellgetriebene Ansätze und entsprechende Beschreibungsmittel besitzen. Die Realisierung und Modifikation der Steuerungssoftware während der Inbetriebnahme und des laufenden Betriebs der Anlage erfolgt aber durch Steuerungsprogrammierer. Sie haben jedoch keine Kenntnisse dieser formalen Beschreibungsmittel und Methoden, aufgrund ihrer beruflichen Ausbildung als Facharbeiter und Techniker. Die notwendigen Kenntnisse über die Anwendung modellgetriebener Ansätze im Steuerungsentwurf müssten folglich durch zusätzliche Schulungen erlangt werden, daher finden formale Methoden entsprechend wenig Anklang bei den Unternehmen. Die in der praktischen Anwendung des Steuerungsentwurfs bevorzugte Vorgehensweise ist aus diesen Gründen die **direkte Implementierung**, bei der die Implementierung auf Grundlage der informellen Spezifikation und ohne eine vorhergehende Formalisierung der Anforderungen erstellt wird.

Typische, in der Praxis des Steuerungsentwurfs angewendete Beschreibungsmittel zur Spezifikation von Steuerungsabläufen, wie beispielsweise Funktionspläne [DIN 40719-6][#] oder Programmablaufgraphen [KILLENBERG 1976], erlauben zwar einen detaillierten Blick auf die Automatisierungslösung, sind aber nur textuell, beispielsweise in Normen oder Richtlinien, beschrieben. Im Falle der in [DIN EN 61131-3][#] definierten Implementierungssprachen, welche im Engineering automatisierter Anlagen als Mittel der Wahl zur Beschreibung von Steuerungsabläufen verwendet werden, ist das Beschreibungsmittel zudem an eine konkrete technische Realisierung gebunden. Nicht selten kommen in der Praxis des Steuerungsentwurfs somit domänenspezifische Beschreibungsmittel zum Einsatz, die hinsichtlich einer formalen Interpretation nicht eindeutig definiert sind. Für die Anwender resultieren daraus Unsicherheiten, die im Rahmen einer manuellen Auswertung zu fehlerhaften Interpretationen und Missverständnissen führen können. Sie werden zumeist erst in nachgelagerten Arbeitsschritten entdeckt und verursachen hohe, zum Teil vermeidbare Kosten. Im Rahmen einer rechnergestützten Handhabung durch Engineering-Werkzeuge führen diese Unsicherheiten zu unterschiedlichen Festlegungen und somit zu herstellerepezifischen Ausprägungen und einer eingeschränkten Interoperabilität [DRATH ET AL. 2011]. Der in diesem Zusammenhang charakteristische Zwiespalt der praktischen Relevanz formaler Beschreibungsmittel mit Hinblick auf einen systematischen Steuerungsentwurf lässt sich, wie in [BARESI ET AL. 2000, S.2437] ausgeführt, wie folgt bewerten:

„Informal approaches better match domain expertise, but can be analyzed only partially, while formal methods may be more difficult to adapt to the problem domain, but can be better analyzed.“

Ein „ideales“ Beschreibungsmittel für einen systematischen Steuerungsentwurf basiert demnach einerseits auf einer mathematisch eindeutigen, formalen Definition und ermöglicht andererseits die technologieneutrale, eingängige und grafische Beschreibung des Steuerungsablaufs als Teil der Automatisierungslösung. Der Abstraktionsgrad der mit dem Beschreibungsmittel erstellten Spezifikation sollte dabei an domänenspezifische Bedürfnisse des Steuerungsentwurfs im Engineering automatisierter Anlagen angepasst und das Beschreibungsmittel sowohl Ingenieuren als auch Facharbeitern und Technikern bekannt sein.

Die vorliegende Arbeit widmet sich, aufgrund der zuvor beschriebenen Rahmenbedingungen für die Einführung formaler Beschreibungsmittel und Methoden im Steuerungsentwurf, dem Beschreibungsmittel *GRAphe Fonctionnel de Commande Etape et Transitions* (GRAFCET), welches gemäß dem internationalen Standard IEC 60848 [IEC 60848 A][#] definiert ist. Inspiriert durch Petrinetze ermöglicht GRAFCET die Spezifikation von Steuerungsabläufen und bietet hierzu eine umfangreiche Anzahl von grafischen Elementen. Mit Ausnahme von Frankreich ist GRAFCET im internationalen Umfeld und auch in Deutschland eher unbekannt und wird oftmals fälschlicherweise mit der SPS-Programmiersprache *Sequential Function Charts* (SFC) gemäß [IEC 61131-3][#] gleichgesetzt, die im deutschen Sprachraum auch unter dem Begriff der *Ablaufsprache* (AS) [DIN EN 61131-3][#] geläufig ist. Allerdings handelt es sich bei GRAFCET um eine Spezifikationssprache und bei SFC um eine Implementierungssprache. GRAFCET [DIN EN 60848][#] hat im Jahr 2002 die zuvor in Deutschland gültige DIN 40719-6 „Regeln für Funktionspläne“ [DIN 40719-6][#] abgelöst und ist mittlerweile fester Bestandteil von durch Industrie- und Handelskammern durchgeführten Zwischen- und Abschlussprüfungen der Ausbildungsberufe *Mechatroniker/-in*, *Elektroniker/-in für Automatisierungstechnik*, *Elektroniker/-in für Betriebstechnik* und *Industriemechaniker/-in* [PAL 2007]. Somit hält GRAFCET Einzug in Lehrpläne für die berufliche Aus- und Weiterbildung zukünftiger potentieller Steuerungsprogrammierer, wie am Beispiel von [TS-HANNOVER 2010, LP-RHEINLAND-PFALZ 2011][@] deutlich wird. Damit wächst in den kommenden Jahren eine neue Generation von Steuerungsprogrammierern heran, die dieses Beschreibungsmittel in ihrer Ausbildung gelernt haben und somit ohne Einstiegshürde in der praktischen Anwendung nutzen können. Aus diesem Sachverhalt ergibt sich die realistische Chance, eine methodische Vorgehensweise für die automatische Generierung von Steuerungscode auf der Basis einer formalen Definition von GRAFCET in der Praxis des Steuerungsentwurfs systematisch einzuführen, welche durch die Anwender akzeptiert wird.

1.2 Wissenschaftlicher Beitrag und Gliederung der Arbeit

Zwar wurde GRAFCET in Anlehnung an Petrinetze konzipiert, allerdings ist derzeit keine formale Definition verfügbar, die alle im Standard IEC 60848 definierten grafischen Elemente beinhaltet. Anknüpfend an ein existierendes, Petrinetz-basiertes formales Modell, welches lediglich einfache GRAFCET-Elemente berücksichtigt, wird im Rahmen dieser Arbeit eine Erweiterung und Vervollständigung der zugehörigen Definitionen vorgeschlagen und diskutiert. Das Hauptaugenmerk richtet sich auf die hierarchischen Konstrukte der **ein-schließenden Schritte** und **zwangssteuernden Befehle** und auf **zeitabhängige Bedingungen**. Eine weitere zentrale Anforderung an eine GRAFCET-Spezifikation im Kontext eines systematischen Steuerungsentwurfs ist, dass die zugehörigen Daten für eine rechnergestützte Handhabung zur Verfügung gestellt werden. Ein hersteller- und werkzeugunabhängiges Datenaustauschformat ist somit notwendig, um einen Zugriff auf die entsprechenden Engineering-Daten zu gewährleisten. Eine entsprechende implementierungsunabhängige Notation für GRAFCET innerhalb eines geeigneten offenen Datenaustauschformats wird daher in dieser Arbeit systematisch erarbeitet und prototypisch implementiert.

Grundsätzliches Ziel der vorliegenden Arbeit ist es, einen Beitrag dazu zu leisten, GRAFCET für die Spezifikation von Steuerungsabläufen im Rahmen eines systematischen Steuerungsentwurfs verwenden zu können, und darauf aufbauend Transformationsalgorithmen zu

entwickeln, mit denen man möglichst alle in IEC 60848 definierten Konstrukte nutzen und in Steuerungscode überführen kann. Aufgrund der weiten Verbreitung von SPSen im Bereich automatisierter Anlagen soll der generierte Steuerungscode den Anforderungen der IEC 61131-3 genügen. Eine wesentliche Grundlage der Transformationsalgorithmen bilden diesbezüglich die Gemeinsamkeiten von GRAFCET und SFCs. Im Rahmen dieser Arbeit werden die den Transformationsalgorithmen zugrunde liegenden Transformationsregeln sowohl konzeptionell als auch in ihrer Umsetzung beschrieben und in einem prototypischen, softwarebasierten Werkzeug implementiert. Insgesamt betrachtet liefern die in dieser Arbeit vorgestellten Ergebnisse bezüglich des Beschreibungsmittels GRAFCET sowie der zugehörigen Methoden und Werkzeuge die Grundlagen für einen systematischen Steuerungsentwurf zur automatischen Generierung von IEC 61131-3 konformem Steuerungscode basierend auf formalen Methoden (→ Abbildung 1-2).

Im Anschluss an dieses einleitende Kapitel folgt in Kapitel 2 zunächst eine nähere Betrachtung der grundsätzlichen Möglichkeiten formaler Methoden im Rahmen eines systematischen Steuerungsentwurfs. Die Analyse wesentlicher, existierender Forschungsansätze zur automatischen Generierung von Steuerungscode erlaubt anschließend eine Bewertung des aktuellen, nicht GRAFCET-spezifischen Forschungsstands. Kapitel 3 erläutert die wesentlichen Eigenschaften der Programmiersprache Sequential Function Charts und stellt gemeinsam mit Kapitel 4, welches GRAFCET im Detail betrachtet, die wesentlichen Gemeinsamkeiten und Unterschiede beider Beschreibungsmittel heraus. Die im Rahmen von Kapitel 5 diskutierten GRAFCET-spezifischen Forschungsansätze runden den Überblick über den aktuellen Forschungsstand ab und verdeutlichen den Handlungsbedarf bezüglich GRAFCET. Im anschließenden Teil dieser Arbeit werden die wesentlichen Erkenntnisse und Ergebnisse des eigenen wissenschaftlichen Beitrags mit dem Ziel eines systematischen

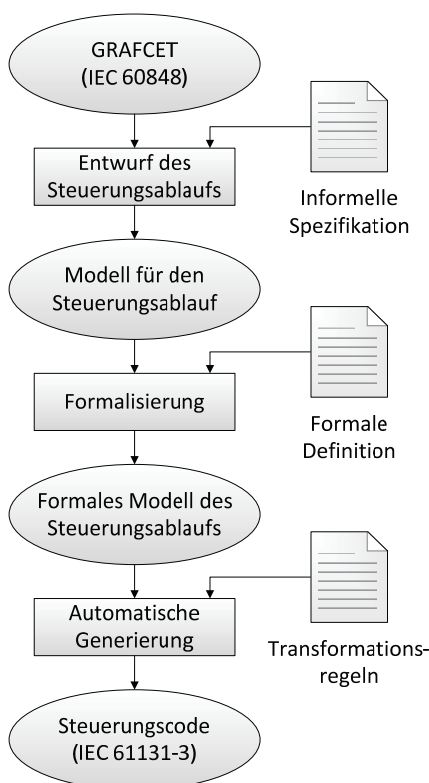


Abbildung 1-2: Systematischer Steuerungsentwurf mit GRAFCET

Steuerungsentwurfs herausgestellt. So reflektiert Kapitel 6 in einem ersten Schritt die formalen Zusammenhänge des Petrinetz-basierten Modells für einfache GRAFCET-Spezifikationen. In einem zweiten Schritt werden, darauf aufbauend, die notwendigen Erweiterungen des formalen Modells festgelegt, so dass GRAFCET dann vollständig als zeitbehaftetes steuerungstechnisch interpretiertes Petrinetz definiert wird.

Daran anknüpfend beschreibt Kapitel 7 die methodischen Grundlagen sowie das Konzept einer implementierungsunabhängigen Notation für GRAFCET in einem offenen XML-basierten Datenaustauschformat, welches als zentrale Datendrehscheibe im systematischen Steuerungsentwurf eingesetzt werden kann. Die methodische Vorgehensweise bei der Transformation von GRAFCET-Spezifikationen in Steuerungscode gemäß IEC 61131-3 wird im anschließenden Kapitel 8 vorgestellt und diskutiert. Für die Umsetzung der zuvor in Kapitel 7 und Kapitel 8 beschriebenen Methoden stellt das Kapitel 9 entsprechende prototypische Werkzeuge

vor, die den Anwender in den einzelnen Arbeitsschritten eines systematischen Steuerungsentwurfs mit GRAFCET unterstützen. In Kapitel 10 wird aufgezeigt, wie ein systematischer Steuerungsentwurf mit GRAFCET unter Anwendung der vorgeschlagenen methodischen Vorgehensweise sowie der prototypisch implementierten Werkzeuge möglich ist. Hierzu werden die einzelnen Arbeitsschritte von der Spezifikation des Steuerungsablaufs mit GRAFCET bis zur Erzeugung IEC 61131-3 konformen Steuerungscode anhand eines industriellen Anlagenbeispiels dargestellt. Eine Bewertung und Zusammenfassung der wesentlichen Erkenntnisse und Ergebnisse in Kapitel 11 rundet die Arbeit ab und mündet in einem Ausblick auf weitere, noch offene Forschungsfragen im Zusammenhang des vorgestellten Ansatzes.

Für eine bessere Lesbarkeit wird in den nachfolgenden Kapiteln unter GRAFCET das grafische Beschreibungsmittel gemäß IEC 60848 im Allgemeinen verstanden. Der Begriff Grafcet bezeichnet eine konkrete Instanz einer GRAFCET-Spezifikation, die mit den grafischen Elementen der IEC 60848 erstellt wurde.

2 Formale Methoden im Steuerungsentwurf

2.1 Steuerungsentwurf in der heutigen Praxis

Unter dem Begriff des Steuerungsentwurfs werden diejenigen Tätigkeiten im Rahmen des Engineerings automatisierter Anlagen zusammengefasst, die notwendig sind, um eine Automatisierungsaufgabe zu analysieren, eine Lösung zu erarbeiten und in ein lauffähiges Steuerungsprogramm umzusetzen. Die Automatisierungsaufgabe selbst sowie die zugehörigen Anforderungen bilden die notwendige Grundlage und werden üblicherweise in textueller Form, wie beispielsweise einem Lastenheft nach [VDI/VDE 3694][#], dokumentiert und durch weitere Planungsdokumente ergänzt. Das Lastenheft, als wesentlicher Teil der informellen Spezifikation, vermittelt im Wesentlichen eine funktionsorientierte Sichtweise auf den durch die Steuerung zu gewährleistenden Sollablauf der automatisierten Anlage. Im Verlauf des Steuerungsentwurfs verändert sich diese funktionsorientierte Sichtweise sukzessive zu einer ereignisdiskreten und schließlich signalorientierten Sichtweise [FISCHER & ENGELL 2011]. Hierbei ist insbesondere der Steuerungsablauf von Bedeutung, da er die generelle Funktionsweise der automatisierten Anlage sowohl für die Planung als auch für die Realisierung beschreibt und als gemeinsames Verständigungsmittel dienen kann. Auf dieser Grundlage erfolgt im Rahmen der Realisierung die Detail-Implementierung eines lauffähigen Steuerungsprogramms, welches beispielsweise durch Verriegelungen, Sicherheits- und weitere Funktionen angereichert wird [GÜTTEL 2012]. Die Programmlogik selbst orientiert sich aber stets am vorgegebenen Steuerungsablauf.

Aus der Perspektive des Steuerungsprogrammierers beginnt der Steuerungsentwurf zunächst damit, die im Lastenheft dokumentierten Anforderungen an die Steuerung zu analysieren und zu interpretieren, um in darauffolgenden Arbeitsschritten die Automatisierungsaufgabe zu lösen. Die Lösung der Automatisierungsaufgabe sowie die Spezifikation des Steuerungsablaufs, wie beispielsweise in [HAJARNAVIS & YOUNG 2008] bestätigt, werden allerdings nicht unter Anwendung eines technologieunabhängigen gewerkespezifischen Beschreibungsmittels entwickelt, sondern im Allgemeinen durch eine direkte Überführung der manuell ausgewerteten Anforderungen in ein Steuerungsprogramm durchgeführt (**direkte Implementierung**). Aufgrund der weiten Verbreitung von SPSen auf dem Gebiet automatisierter Anlagen erfolgt die direkte Implementierung in einer der zugehörigen, in [DIN EN 61131-3][#] definierten Programmiersprachen. Vor allem die textbasierten Programmiersprachen *Anweisungsliste* (AWL) und *Strukturierter Text* (ST) sowie die grafische Programmiersprache *KontaktPlan* (KOP), die optisch an die Relais-Schaltungstechnik erinnert, erfreuen sich in der praktischen Anwendung großer Beliebtheit, wie in [LUCAS & TILBURY 2004] oder [LJUNGKRANTZ & AKESSON 2007] herausgestellt wird. SFCs zur grafischen Beschreibung des Ablaufs des Steuerungsprogramms werden demnach nur selten verwendet.

Ein wesentlicher Arbeitsschritt des systematischen Steuerungsentwurfs wird in der heutigen praktischen Anwendung allerdings übersprungen: die Formalisierung der Spezifikation. Als Folgen ergeben sich aus der praktizierten direkten Implementierung die nachfolgend aufgeführten Nachteile:

- Durch eine textuelle Formulierung von Anforderungen kann nicht immer sicher-gestellt werden, dass alle am Steuerungsentwurf Beteiligten ein ein-

heitliches Verständnis der Anforderungen an das Automatisierungssystem besitzen, da Text als Beschreibungsmittel nur informellen Charakter besitzt.

- Die Vorgehensweise der direkten Implementierung ist in starkem Maße von den Erfahrungen des Steuerungsprogrammierers bzw. der Steuerungsprogrammierer abhängig.
- Die manuelle Auswertung und direkte Implementierung der Anforderungen führt, insbesondere durch die zunehmende Komplexität automatisierter Anlagen, zu Fehlern, die sich in ungewünschtem Verhalten der Anlage auswirken. Diese Fehler werden in den meisten Fällen erst in späteren Engineering-Phasen entdeckt, wie beispielsweise der Phase der Inbetriebnahme, und ziehen langwierige Korrekturen nach sich [TAUCHNITZ 2013]. Vereinzelt durchgeführte Tests des Steuerungsprogramms leisten lediglich eine stichprobenartige Überprüfung.
- Die Implementierung ist die einzige Dokumentation des Steuerungsalgorithmus. Ein gewerkeübergreifendes Verständnis über den Steuerungsablauf ist somit nicht möglich. Es erfordert hohen Aufwand, um die Logik des Steuerungsalgorithmus nachzuvollziehen und Änderungen vornehmen zu können.

Insgesamt betrachtet weist die heutige Vorgehensweise eine Vielzahl von Schwachstellen auf. Für eine ausführliche Analyse und Diskussion der charakteristischen Eigenschaften des heutigen Steuerungsentwurfs am Beispiel verfahrenstechnischer Anlagen sei an dieser Stelle auf [LOHMANN 2011] verwiesen. In den nachfolgenden Kapiteln wird überblicksartig aufgezeigt, wie dieser fehlerbehafteten Vorgehensweise grundsätzlich (→ 2.2) und im Detail (→ 2.3) mit formalen Methoden begegnet wird, um einen systematischen Steuerungsentwurf im Sinne eines MDE zu ermöglichen und den Anteil nicht wertschöpfender Tätigkeiten zu reduzieren. Der Fokus liegt hierbei, wie bereits beschrieben, auf geeigneten Beschreibungsmitteln zur Spezifikation des Steuerungsablaufs.

2.2 Steuerungsentwurf auf der Basis grafischer Beschreibungsmittel

2.2.1 Grundlagen

Auch in einem systematischen Steuerungsentwurf bilden die Anforderungen an das Automatisierungssystem sowie an das zu steuernde System den Ausgangspunkt aller nachfolgenden Tätigkeiten und später die Grundlage für die Überprüfung der Automatisierungslösung. Im Gegensatz zur heute üblichen Praxis erfolgt eine Implementierung beim systematischen Steuerungsentwurf erst nach der Erstellung einer Spezifikation, die als technologieunabhängiges Modell des Steuerungsablaufs angesehen werden kann. Das Hauptaugenmerk des Steuerungsentwurfs sollte auf der Spezifikation liegen, da im Rahmen dieser Modellierung die eigentlich kreative Arbeit geleistet wird. Die Implementierung, so das Ziel, wird durch rechnergestützte Transformation (teil-) automatisch aus der Spezifikation abgeleitet. Eine wesentliche Voraussetzung für die Adaption entsprechender Methoden der Informatik ist, dass die Modellinformationen für softwaregestützte Werkzeuge zugänglich sind und durch Algorithmen verarbeitet werden können. Die zugehörigen grafischen Be-

schreibungsmittel müssen dazu einer eindeutigen, mathematischen Definition unterliegen, die im Rahmen dieser Arbeit unter dem Begriff des **formalen Modells** zusammengefasst wird.

Unter einem **formalen Beschreibungsmittel** wird im Kontext dieser Arbeit ein Beschreibungsmittel verstanden, welches in mathematisch eindeutiger Art und Weise bezüglich seiner Struktur und seines dynamischen Verhaltens definiert ist, mit dem daher formale Modelle erstellt werden können und das somit durch rechnergestützte Methoden eindeutig analysiert und interpretiert werden kann. Ist das Beschreibungsmittel nicht durch einen mathematischen Formalismus definiert, sondern nur textuell, wie dieses beispielsweise in Normen oder Richtlinien oftmals der Fall ist, so wird es als **semi-formales Beschreibungsmittel** bezeichnet. Auch semi-formale Beschreibungsmittel können mit Hilfe rechnergestützter Methoden analysiert und interpretiert werden. Aufgrund ihrer textuellen Definition und der damit möglichen Verständnisunterschiede können in Detailfragen allerdings verschiedene Interpretationen der Struktur oder des dynamischen Verhaltens zulässig sein. Zugehörige Methoden sind folglich nicht direkt übertragbar und allgemeingültig für das semi-formale Beschreibungsmittel, sondern von der individuellen Sichtweise auf das jeweilige Beschreibungsmittel abhängig und werkzeugspezifisch. Zur Anwendung allgemeingültiger, formaler Methoden muss ein semi-formales zuerst in ein formales Beschreibungsmittel überführt werden. Dies geschieht entweder durch Integration in ein bestehendes formales Modell oder durch die Definition eines eigenständigen Modells. Sollte ein Beschreibungsmittel weder mathematisch eindeutig noch durch Konsens allgemeingültig definiert sein, so handelt es sich um ein **informelles Beschreibungsmittel**. Das am weitesten verbreitete und im Steuerungsentwurf genutzte informelle Beschreibungsmittel „[...] *ist heute noch die jeweils nationale Umgangssprache.*“ [EPPL 2003, S. 64]. Der systematische Steuerungsentwurf auf der Basis eines formalen Beschreibungsmittels eröffnet somit grundsätzlich die Möglichkeiten, die Spezifikation des Steuerungsablaufs (*Modell der Steuerung*) mittels rechnergestützter Methoden zu verifizieren und zu testen, das Modell der Steuerung automatisch aus den Anforderungen abzuleiten (*Synthese*) und die Implementierung in Form von Steuerungscode automatisch zu generieren (→ Abbildung 2-1). Insbesondere durch die konsequente Anwendung formaler Methoden zur automatischen Generierung von Steuerungscode gelingt es, qualitativ hochwertigen Steuerungscode zu generieren, da eine rechnergestützte Handhabung eindeutiger Transformationsvorschriften menschliche Interpretationsfehler bei der Implementierung ausschließt.

Das *Modell der Steuerung* bildet den zentralen Aspekt zur Anwendung formaler Methoden (→ Abbildung 2-1). Die darauf abgestimmten Vorgehensweisen zu *Verifikation und Test*, zur *Steuerungssynthese* sowie zur *automatischen Generierung* werden in den nachfolgenden Kapiteln erläutert und tragen zur Einordnung und Abgrenzung des Inhalts dieser Arbeit bei. Alle Vorgehensweisen unterliegen der Grundannahme, dass zumindest die benötigten Modelle in geeigneter formaler Form vorliegen. Der Fokus der hier vorliegenden Arbeit ist in Abbildung 2-1 hervorgehoben und wird im Rahmen des Zwischenfazit (→ 2.4) noch einmal aufgegriffen. Eine ausführliche Betrachtung formaler Methoden im Steuerungsentwurf findet sich beispielsweise in [FREY & LITZ 2000].

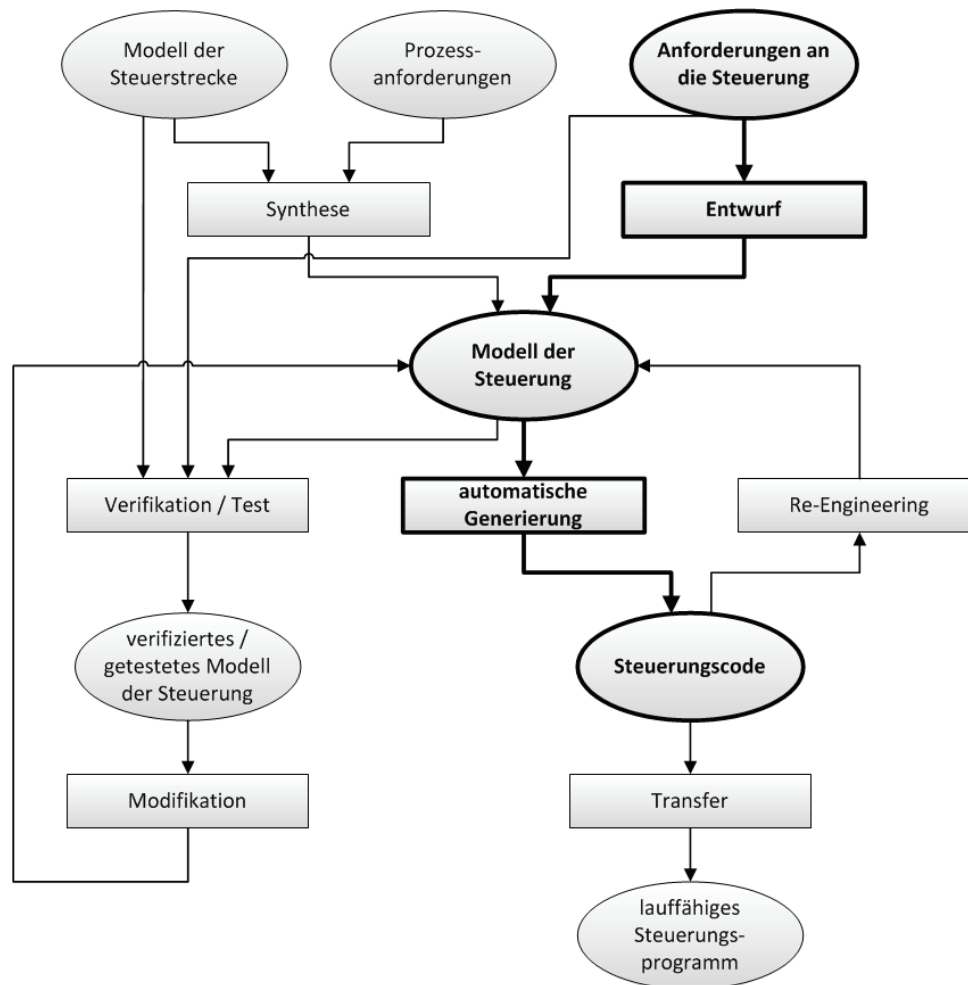


Abbildung 2-1: Prinzipielle Möglichkeiten formaler Methoden im Steuerungsentwurf, in Anlehnung an [KOWALEWSKI 1996, S. 3]

2.2.2 Verifikation und Test

Zielsetzung der formalen *Verifikation* ist die systematische und möglichst vollständige Überprüfung der Korrektheit eines Modells [LUNZE 2006]. Die Untersuchung des Modells erfolgt durch den Vergleich des Modellverhaltens mit dem spezifizierten Sollverhalten anhand unterscheidbarer Zustände, wie beispielsweise der Nachweis der grundlegenden strukturellen Eigenschaften *Erreichbarkeit* oder *Lebendigkeit*. Hinsichtlich der Verifikation des Modells der Steuerung muss insbesondere das Modell des ungesteuerten technischen Systems, der *Steuerstrecke*, berücksichtigt werden, um verwertbare Ergebnisse zu erzielen. Durch die Verknüpfung beider Modelle über Eingaben und Ausgaben existieren Wechselwirkungen, die das Sollverhalten maßgeblich beeinflussen. Doch die zu analysierende Anzahl von Zuständen steigt insbesondere bei einer Vielzahl von Eingangs- und Ausgangssignalen sowie zeitabhängigen Bedingungen exponentiell an und ist ein wesentlicher Grund für andauernde Forschungsaktivitäten, wie beispielsweise die Diskussion der Notwendigkeit eines geeigneten Anlagenmodells zur Begrenzung des Zustandsraums [MACHADO ET AL. 2006]. In den meisten Fällen wird der Zustandsraum formal auf endliche Zustandsautomaten oder Petrinetze zurückgeführt, wie beispielsweise in [BARRANGAN SANTIAGO & FAURE 2005] oder [MERTKE & FREY 2001] aufgezeigt, da für diese Beschreibungsmittel ausgereifte Werkzeuge für eine rechnergestützte Handhabung zur Verfügung stehen. Ein alternativer Ansatz

auf der Basis algebraischer Gleichungen wird beispielsweise in [ROUSSEL & FAURE 2002] vorgestellt. Weiterführende Untersuchungen bezüglich des dynamischen Verhaltens des Steuerungsmodells werden im Zusammenhang mit einem Modell der Steuerstrecke durchgeführt. Die wesentliche zu beantwortende Frage ist, inwiefern die Anforderungen an das Sollverhalten des technischen Systems durch das Modell der Steuerung erfüllt werden. Diese oftmals auch mit dem Begriff der *Validierung* benannten Tests können durch eine geeignete Simulation unterstützt werden und sollen das Verhalten des gesteuerten technischen Systems möglichst vollständig abdecken. Neben den formalen Modellen der Steuerstrecke und Steuerung ist zur Durchführung der Tests zusätzlich eine entsprechend formale Beschreibung der Anforderungen erforderlich, um geeignete Testsequenzen festlegen zu können. [PROVOST ET AL. 2009] stellen diesbezüglich eine Methode vor, wie solche Testsequenzen automatisch generiert werden können. Dennoch ist die Formalisierung der Anforderungen ein wesentlicher Engpass für eine formale Validierung durch Test [LJUNGKRANTZ ET AL. 2010]. Auch hier werden als Modelle oftmals endliche Zustandsautomaten oder Petrinetze zugrunde gelegt, wie beispielsweise in [FREY & LITZ 1998] oder [KLEIN ET AL. 2002] vorgeschlagen.

Insgesamt betrachtet bieten formale Verifikation und Validierung durch Test gute Ansatzpunkte für eine Überprüfung der wesentlichen Zusammenhänge zwischen Steuerstrecke und Steuerung im Zusammenhang eines systematischen Steuerungsentwurfs. So können Fehler bereits auf Modellebene identifiziert und in der Entwurfsphase beseitigt werden, bevor eine Implementierung erfolgt. Die Aussagekraft der Analysen ist aber unmittelbar vom Umfang der formalen Modelle abhängig und im Falle der Testsequenzen nur schwer auf Vollständigkeit überprüfbar, da die Anforderungen ursprünglich nur textuell, beispielsweise in einem Lastenheft, beschrieben worden sind. Zudem muss der zusätzliche Aufwand im Steuerungsentwurfsprozess berücksichtigt werden, welcher sich ohne die Möglichkeit einer automatischen Generierung in der Praxis kaum rechtfertigen lässt.

2.2.3 Steuerungssynthese

Im Gegensatz zur Verifikation und Validierung durch Test verfolgt die Steuerungssynthese für den Bereich der formalen Methoden einen grundsätzlich anderen Weg und zielt auf die automatische Berechnung des Steuerungsmodells ab. Ausgangspunkt bildet das formale Modell der Steuerstrecke. Unter Einbeziehung von Anforderungen und Restriktionen hinsichtlich des gewünschten dynamischen Verhaltens (**Prozessanforderungen**) kann das Streckenverhalten gemäß dem Steuerungsziel eingeschränkt und anschließend ein entsprechendes Steuerungsmodell abgeleitet werden. Dies wird am Beispiel von Petrinetzen in [DIDEBAN & ALLA 2006] aufgezeigt. Ähnlich zur Verifikation und Validierung durch Test werden auch für die Steuerungssynthese möglichst detaillierte formale Modelle der Steuerstrecke sowie der Prozessanforderungen benötigt.

Eine für den Bereich der Steuerungssynthese bedeutende methodische Vorgehensweise stellt die sogenannte *Supervisory Control Theory* (SCT) nach [RAMADGE & WONHAM 1986] dar. Aus dieser Methode zur Steuerungssynthese resultiert ein übergeordnetes Steuerungsmodell (*Supervisor*) in Form eines endlichen Zustandsautomaten, welches das ungesteuerte Verhalten der Steuerstrecke derart einschränkt, dass das durch die Prozessanforderungen beschriebene Steuerungsziel erreicht wird. Das Modell des Supervisors basiert allerdings auf der Annahme, dass die Steuerstrecke, beispielsweise eine Anlage, spontane Ereignisse generiert. Zudem

kann der Supervisor die Steuerstrecke definitionsgemäß nicht in einen bestimmten Zustand zwingen. Dies stellt gleichzeitig die wesentliche Einschränkung dieser Methode dar. Auf der SCT aufbauende Methoden, wie beispielsweise [CHARBONNIER ET AL. 1995], fokussieren deshalb auf diese grundlegenden Nachteile und diskutieren entsprechende Erweiterungen dieses Ansatzes für den Bereich ereignisdiskreter Systeme. Eine Methode für verteilte Automatisierungssysteme wird darüber hinaus in [STURSBURG 2012] vorgeschlagen. Weitere Ansätze, wie beispielsweise [ROUSSEL ET AL. 2004] oder [HIETTER ET AL. 2008], verwenden anstatt endlicher Zustandsautomaten algebraische Gleichungen zur Modellierung der Steuerstrecke und der Prozessanforderungen.

Die wesentliche Hürde bei der Steuerungssynthese besteht nach wie vor darin, ein detailliertes Modell der Steuerstrecke zu erstellen, wobei auch Fehlerzustände und theoretisch mögliche Zustände der ungesteuerten Strecke mit einbezogen werden müssen. Dadurch ist diese methodische Vorgehensweise bezüglich eines systematischen Steuerungsentwurfs grundsätzlich fehleranfällig, wenn das Modell der Steuerstrecke manuell erstellt werden muss. Aus diesem Grund ist die Steuerungssynthese weiterhin Gegenstand zahlreicher Forschungsansätze und derzeit lediglich für theoretische Betrachtungen geeignet.

2.2.4 Automatische Generierung

Die Methode der automatischen Generierung kann als integraler Bestandteil eines systematischen Steuerungsentwurfs im Sinne des MDEs angesehen und in zwei Bestandteile untergliedert werden. Hierzu zählt einerseits die automatische Generierung der Implementierung in Form von Steuerungscode und andererseits die automatische Generierung eines Steuerungsmodells auf Basis der Implementierung. In beiden Fällen führt eine rechnergestützte Transformation zur Überführung des Ausgangsmodells in das jeweilige Zielmodell unter Berücksichtigung der zugehörigen Transformationsregeln. Die automatische Generierung der Implementierung ist hauptsächlich auf einen vorwärts gerichteten Steuerungsentwurf bezogen, der schrittweise von der Anforderungsanalyse über den Entwurf zur Implementierung verläuft. Als Implementierungsplattformen kommen sowohl Mikrocontroller [DEZANI ET AL. 2011] als auch SPSen [ESTÉVEZ ET AL. 2007] oder weitere Steuerungssysteme in Frage. Aufgrund der weiten Verbreitung von SPSen im Bereich automatisierter Anlagen liegt das Hauptaugenmerk im Rahmen dieser Arbeit auf der automatischen Generierung IEC 61131-3 konformen SteuerungsCodes. Dies wird durch die Kennzeichnung einer vorwärtsgerichteten Vorgehensweise in Abbildung 2-1 verdeutlicht. Die entgegengesetzte automatische Generierung des Steuerungsmodells unterstützt vorrangig einen rückwärtsgerichteten Steuerungsentwurf, wie beispielsweise in [WIGHTKIN ET AL 2011] und [KABRA ET AL. 2012] diskutiert, und bildet die Grundlage einer effizienten Analyse bereits existierender Steuerungsprogramme [SARMENTO ET AL. 2008]. Dieser Ansatz wird auch als *Re-Engineering* bezeichnet [BANI YOUNIS 2006]. Das Hauptaugenmerk dieser Arbeit liegt auf der Schnittstelle zwischen Planung und Realisierung im Steuerungsentwurf automatisierter Anlagen, so dass die vorwärtsgerichtete Vorgehensweise den Regelfall darstellt (→ 2.1). Aus diesem Grund wird ein *Re-Engineering* nicht weiter betrachtet.

Berücksichtigt man die heutige informelle Vorgehensweise im Steuerungsentwurf, so ermöglichen Methoden zur automatischen Generierung den Zugang zu einem rechnergestützten gewerkeübergreifenden Datenaustausch und unterstützen somit einen durch-

gängigen Arbeitsablauf. Die Bereitstellung solcher Schnittstellen zur automatischen Generierung kann beispielsweise auf der Grundlage von Transformationsregeln erfolgen und unmittelbar in heutige Arbeitsabläufe integriert werden. Alle weiterführenden formalen Methoden, deren Ziel und Zweck in den vorangegangenen Unterabschnitten skizziert wurde, bedürfen hingegen noch weiterer Forschungsaktivitäten und einer geeigneten Einführungsstrategie in die Praxis des Engineerings automatisierter Anlagen. Das nachfolgende Kapitel gibt einen Überblick über den aktuellen Forschungsstand bezüglich der automatischen Generierung IEC 61131-3 konformen Steuerungscode ausgehend von der Spezifikation des Steuerungsablaufs mit Hilfe von grafischen Beschreibungsmitteln. Die grafischen Beschreibungsmittel werden in domänenunabhängige Beschreibungsmittel, deren Anwendungsbereich nicht auf die spezifischen Belange der Automatisierungstechnik zugeschnitten ist, und in für die Domäne der Automatisierungstechnik spezifische Beschreibungsmittel unterschieden.

2.3 Forschungsansätze zur automatischen Generierung

2.3.1 Domänenunabhängige Beschreibungsmittel

2.3.1.1 *Unified Modeling Language*

In heutigen Forschungsansätzen werden zunehmend Beschreibungsmittel für einen systematischen Steuerungsentwurf betrachtet, die ihren Ursprung in der Informatik haben und einen objektorientierten Softwareentwurf unterstützen, wie beispielsweise die *Unified Modeling Language* (UML) [OMG UML, v2.4.1][#] in [WITSCH ET AL. 2008]. Bei der UML handelt es sich um ein semi-formales Beschreibungsmittel, welches es erlaubt, eine Software aus unterschiedlichen Perspektiven zu spezifizieren [KATZKE & VOGEL-HEUSER 2009]. Für jede Perspektive steht ein bestimmtes UML-Diagramm zur Verfügung, wobei hinsichtlich der Spezifikation dynamischer Aspekte insbesondere *UML-Aktivitätsdiagramme* in Betracht gezogen werden. Die grundsätzliche Idee, objektorientierte Methoden in den Steuerungsentwurf automatisierter Anlagen zu übernehmen, spiegelt sich nicht zuletzt in den Bestrebungen wider, den internationalen Standard IEC 61131-3 für SPS-Programmiersprachen in seiner neuesten Version um geeignete objektorientierte Konstrukte bezüglich der Funktionsbausteine zu erweitern [IEC 61131-3][#]. Somit lässt sich die in der Informatik etablierte Vorgehensweise eines objektorientierten Entwurfsansatzes mit Hilfe eines spezifischen UML-Sprachderivats [KATZKE & VOGEL-HEUSER 2007] auch an Automatisierungstechnische Anforderungen anpassen und eine enge Kopplung zu einer IEC 61131-3 konformen Implementierung etablieren. Durch die einzelnen Diagramme der UML werden somit die strukturellen und dynamischen Zusammenhänge des Steuerungsprogramms grafisch aufbereitet und in ihrem Gesamtzusammenhang veranschaulicht. Weiterführend können IEC 61131-3 konforme Programmkonstrukte aus den einzelnen UML-Diagrammen direkt abgeleitet werden, da sie auf demselben Datenbestand basieren. Eine prototypische und werkzeugspezifische Implementierung für einen solchen integrierten Ansatz wird beispielsweise in [WITSCH & VOGEL-HEUSER 2009] vorgestellt.

2.3.1.2 *Statecharts*

Ein weiterer Ansatz für die automatische Generierung von Steuerungscode basiert auf dem grafischen Beschreibungsmittel der *Statecharts*. Statecharts sind gemäß [HAREL & PNUELI 1985] abgeleitet von endlichen Zustandsautomaten und erlauben die grafische Modellierung

reaktiver Systeme in unterschiedlichen Bereichen der Informatik und unter Einbeziehung hierarchischer Strukturen sowie zeitabhängiger Bedingungen. Aufgrund ihres semi-formalen Charakters existieren viele unterschiedliche Interpretationen von Statecharts [VON DER BEECK 1994], wobei die durch David Harel in [HAREL 1987] gegebenen Definitionen aus heutiger Sicht als nahezu allgemeingültig angesehen werden können. Zusätzlich sind Statecharts ein Teil der UML, so dass der im vorangegangenen Unterabschnitt erläuterte Ansatz auch dieses grafische Beschreibungsmittel umfasst [WITSCH ET AL. 2010]. Hinsichtlich eines systematischen Steuerungsentwurfs wären Statecharts demnach zur Spezifikation von Steuerungsabläufen geeignet und können, beispielsweise gemäß der in [MACHADO ET AL. 2001] vorgeschlagenen Methode, in IEC 61131-3 konforme Programmkonstrukte überführt werden. Insbesondere die Ähnlichkeit zu dem Sprachkonzept von SFCs bietet einen Ansatzpunkt für weiterführende formale Methoden, wie im Rahmen einer Gegenüberstellung in [BAUER 2002] verdeutlicht wird. Aufbauend auf diesen Betrachtungen stellen [BAYRAK ET AL. 2008] ein Konzept sowie eine prototypische Implementierung für die automatische Transformation von Statecharts in eine werkzeugspezifische Interpretation von SFCs vor. Die Modellierung der Statecharts erfolgt in diesem Falle mit Hilfe des softwarebasierten Werkzeugs *Stateflow* [STATEFLOW][®], welches in die Entwicklungsumgebung von *MATLAB/Simulink* [MATLAB/SIMULINK][®] integriert ist. Ein zugehöriger, externer Codegenerator sorgt für die Transformation. Einen ähnlichen Ansatz für die systematische Überführung von Statecharts in IEC 61131-3 konforme SFCs schlagen beispielsweise [VIDANAPATHIRANA ET AL. 2011] vor.

2.3.1.3 *Coloured Petri Nets*

Ähnlich zu Statecharts (→ 2.3.1.2) existiert auch im Bereich der Petrinetze eine Vielzahl unterschiedlicher, anwendungsfallspezifischer Petrinetz-Varianten, die sich aus formaler Sicht einer der grundlegenden Petrinetz-Klassen zuordnen lassen. *Coloured Petri Nets* (CPN), sogenannte farbige Petrinetze, gehören zur Klasse der sogenannten *High-Level-Petrinetze* [ISO/IEC 15909-1][#] und stellen ein formales Beschreibungsmittel aus der Familie der Petrinetze dar. Ihr grundlegendes Konzept sowie zugehörige Methoden, Anwendungsfälle und die formale Definition sind beispielsweise in [JENSEN 1997 A-C] ausführlich erläutert. Demnach handelt es sich bei CPN vorrangig um ein auf Petrinetzen basierendes Beschreibungsmittel zur Spezifikation, Simulation, Validierung und Implementierung komplexer Softwaresysteme aus dem Bereich der Informatik [JENSEN 1997 A]. Allerdings gibt es keine ausdrückliche Einschränkung der Anwendungsmöglichkeit, so dass auch eine Integration in den Steuerungsentwurf automatisierter Anlagen denkbar ist und die Nutzung existierender formaler Methoden, wie beispielsweise zur Verifikation, angedacht werden kann. Ein entsprechender Ansatz für den systematischen Steuerungsentwurf auf der Basis von CPN und unter automatischer Generierung von Steuerungscode gemäß IEC 61131-3 wird in [FELDMANN ET AL. 1999 A, FELDMANN ET AL. 1999 B] vorgeschlagen. Ausgehend von dem Modell des Steuerungsablaufs als CPN unterstützt eine zugehörige textuelle Beschreibung die Compiler-basierte Transformation in Steuerungsprogramme, welche den Vorgaben der Programmiersprache ST entsprechen. Die automatische Generierung fokussiert sich ausschließlich auf die Programmlogik innerhalb eines Steuerungsprogramms, so dass weitere Arbeitsschritte, wie beispielsweise die Hardware-Konfiguration, notwendig sind, um ein lauffähiges Steuerungsprogramm zu erhalten.

2.3.2 Domänenspezifische Beschreibungsmittel

2.3.2.1 *Net Condition / Event Systems*

In Anlehnung an das Paradigma der SCT stellen *Condition / Event Systems* (CES) nach [SREENIVAS & KROGH 1991] ein modulares Konzept zur Spezifikation ereignisdiskreter Systeme dar, deren Abhängigkeiten in Form von Bedingungen und Ereignissen beschrieben werden können. Die davon abgeleiteten und formal auf *Stellen-/ Transitions-Netzen* (ST-Netz) basierenden *Net Condition / Event Systems* (NCES) [RAUSCH & HANISCH 1995] stellen ein auf die spezifischen Anforderungen des Steuerungsentwurfs angepasstes formales Beschreibungsmittel dar, welches auf eine signalorientierte Sichtweise fokussiert. Dabei wird das Systemverhalten in einzelne Module untergliedert, deren Verhalten jeweils durch NCES beschrieben und deren Wechselwirkungen und Abhängigkeiten durch Bedingungssignale und Ereignissignale modelliert werden. Die mit Hilfe von NCES erstellte Spezifikation beschreibt dann entweder das Verhalten der ungesteuerten Anlage bzw. des ungesteuerten Prozesses, wie beispielsweise in [HANISCH ET AL. 1998] diskutiert, oder das Verhalten der Steuerung, wie am Beispiel von SPSen in [HANISCH ET AL. 1997] erläutert. Ein Ansatz zur automatischen Generierung IEC 61131-3 konformen Steuerungscode wird in [THIEME & HANISCH 2002] vorgeschlagen. Hier wird sowohl das ungesteuerte Verhalten der Anlage als auch das Verhalten der Steuerung jeweils in einem NCES modelliert. Diese methodische Vorgehensweise ermöglicht das Testen der Steuerung gegen das Anlagenverhalten. Einzelne Funktionen der Steuerung werden in Modulen gekapselt und durch Ereignisse und Bedingungen in den gesamten Steuerungsablauf eingebettet. Die hierarchischen Verknüpfungen der einzelnen NCES-Module können bei der anschließenden Transformation unmittelbar in eine entsprechende Funktionsbausteinstruktur gemäß IEC 61131-3 überführt werden. Auf oberster Hierarchieebene entspricht das NCES dem Hauptprogramm, in dem die Programmvariablen deklariert und aus dem die einzelnen Funktionsbausteinaufrufe koordiniert werden. Als Zielsprachen der IEC 61131-3 werden sowohl AWL als auch ST angenommen.

2.3.2.2 *Steuerungstechnisch interpretierte Petrinetze*

Durch die Anwendung von Petrinetzen im Steuerungsentwurf kann, nicht zuletzt aufgrund der mathematischen Basis der Petrinetze, auf eine Vielzahl von bereits existierenden formalen Methoden und Werkzeugen zurückgegriffen werden, wie am Beispiel von *Steuerungstechnisch Interpretierten PetriNetzen* (SIPN) in [KLEIN ET AL. 2002] deutlich wird. Bei SIPN handelt es sich in diesem Fall um ein auf *Bedingungs-/ Ereignis-Netzen* (BE-Netz) basierendes formales Beschreibungsmittel, welches den Steuerungsablauf aus einer signalorientierten Sichtweise beschreibt und die Spezifikation Boolescher Eingangs- und Ausgangssignale erlaubt. Auf der Basis dieser SIPN wird in [FREY & SCHMIDT 2000, FREY 2000] eine Methode für die automatische Generierung von IEC 61131-3 konformen Steuerungsprogrammen vorgeschlagen, welche in den Programmiersprachen AWL und KOP implementiert werden. Ein softwarebasiertes Werkzeug, dessen prototypische Implementierung in [MINAS & FREY 2002] vorgestellt wird, ermöglicht die grafische Spezifikation des Steuerungsablaufs als SIPN und stellt die entsprechenden Funktionalitäten zur formalen Analyse und automatischen Generierung zur Verfügung. Eine ausführliche Dokumentation des gesamten Ansatzes findet sich in [FREY 2002].

Neben diesem Ansatz zum systematischen Steuerungsentwurf mit SIPN existieren weitere formale SIPN-Modelle, die eine andere Perspektive auf das zu beschreibende Automatisierungssystem vermitteln, wie beispielsweise die auf ST-Netzen basierenden SIPN in [DAVID & ALLA 1992]. Daraus ergeben sich leichte Unterschiede hinsichtlich des Abstraktionsgrades, des Beschreibungsumfangs sowie der anwendbaren formalen Methoden. Insgesamt betrachtet und trotz der Vorzüge eines grafisch eingängigen, petrinetzbasierten Steuerungsentwurfs variieren die einzelnen SIPN-Modelle im Detail. Ein vereinheitlichtes, standardisiertes SIPN-Modell ist derzeit nicht bekannt, so dass kein allgemeingültiger, in der praktischen Anwendung akzeptierter Ansatz existiert.

2.3.2.3 *Dependency Charts / Function Table*

Eine Methode zur Unterstützung des Steuerungsentwurfes unter Anwendung eines neu definierten Beschreibungsmittels wird in [LOHMANN ET AL. 2007] vorgeschlagen. Die Autoren definieren darin ein formales Beschreibungsmittel *Dependency Charts / Function Table* (DC/FT), welches zu einer systematischen Umsetzung der textuell spezifizierten Anforderungen an den Steuerungsalgorithmus und die Anlage in ein lauffähiges SPS-Programm beiträgt. DC/FT ist unmittelbar an der heutigen Arbeitsweise im Steuerungsentwurf ausgerichtet, welche beispielsweise durch eine schrittweise Anreicherung und Revisionierung der Daten bezüglich des Steuerungsprogramms gekennzeichnet ist. Dabei wird die Struktur des Steuerungsprogramms in grafischer Form durch die *Dependency Charts* beschrieben, wobei die einzelnen Zustände und Zustandsübergänge in einen zeitlichen Zusammenhang gesetzt werden. Weiterführende Zusammenhänge bezüglich des Steuerungsprogramms, wie beispielsweise Variablendeklarationen, werden textuell und in tabellarischer Form in den sogenannten *Function Tables* erfasst. Mit Hilfe eines in [FISCHER & ENGELL 2011] vorgestellten Software-Werkzeuges wird sowohl die Vorgehensweise nach dem *top-down*- als auch nach dem *bottom-up*-Prinzip durch Dekomposition und Komposition unterstützt. Das Beschreibungsmittel DC/FT lässt sich somit in einen systematischen Steuerungsentwurf integrieren, wobei das resultierende Steuerungsprogramm automatisch in Form eines SFC generiert wird [LOHMANN 2011]. Die entsprechenden Transformationsregeln erlauben eine unidirektionale Übersetzung von DC/FT in SFC.

2.3.2.4 *Formalisierte Prozessbeschreibung*

In den bisher vorgestellten Ansätzen zur automatischen Generierung von Steuerungscode auf der Grundlage grafischer Beschreibungsmittel stellt die Erstellung eines Modells für den Steuerungsablauf den zentralen Aspekt dar (→ Abbildung 2-1). Im Gegensatz dazu stellt die *Formalisierte Prozessbeschreibung* [VDI/VDE 3682][#] den in einer Anlage ablaufenden technischen Prozess in den Vordergrund der Spezifikation. Die Formalisierte Prozessbeschreibung ist ein grafisches formales Beschreibungsmittel, welches „[...] den am Engineering beteiligten Fachdisziplinen ein eindeutiges und gemeinsames Prozessverständnis ermöglichen“ [CHRISTIANSEN ET AL. 2012, S. 1]^{*} soll und eine wertvolle Ergänzung einer Anlagenstrukturbeschreibung darstellt [JÄGER ET AL. 2012]. Die eindeutige grafische Repräsentation in Verbindung mit einem hinterlegten Informationsmodell bildet das Fundament für eine durchgängige rechnergestützte Handhabung. Es können sowohl verfahrenstechnische als auch fertigungstechnische Prozesse in einer *top-down* oder *bottom-up* Vorgehensweise spezifiziert werden, wobei die unterste Dekompositionsebene Signale aus der Sensor-/ Aktor-Ebene berücksichtigen kann. Aus der Perspektive des technischen

Prozesses können somit einzelne Steuerbefehle und auch Sequenzen in einer Formalisierten Prozessbeschreibung modelliert werden. Die daraus resultierende Ablaufbeschreibung ermöglicht es, Steuerungscode automatisch aus einer Formalisierten Prozessbeschreibung abzuleiten, wie in [ULRICH ET AL. 2009, S.86] beschrieben:

„Durch die Detaillierung von Grundfunktionen bis auf die Ebene der einzelnen Steuerbefehle und die Bezüge zu den PLT-Stellen der technischen Ressource [...] ist es [...] möglich, das Steuerungsprogramm für die automatische Durchführung des Prozesses in Form eines SFC [...] automatisch zu generieren.“

Ein weiterer Ansatz zur automatischen Generierung von Steuerungscode, unter Einbeziehung einer Prozessbeschreibung gemäß VDI/VDE-Richtlinie 3682, auf der Basis wissensbasierter Methoden wird in [GÜTTEL ET AL. 2008, GÜTTEL 2012] vorgeschlagen.

2.4 Zwischenfazit

Alle bisher diskutierten Forschungsansätze zur automatischen Generierung (→ 2.3) haben gemeinsam, dass ihre praktische Anwendung detaillierte Kenntnisse des jeweiligen grafischen Beschreibungsmittels erfordert. Die Akzeptanz für solche, oftmals aus dem Bereich der Informatik stammenden formalen Ansätze zur Softwareentwicklung ist nicht zuletzt deswegen in der Praxis sehr gering. Eine mögliche Ursache hierfür ist, dass Steuerungsprogrammierer aufgrund ihres beruflichen Werdegangs zumeist keine Kenntnisse dieser semi-formalen bzw. formalen Beschreibungsmittel und Methoden besitzen, so dass die entsprechenden Forschungsansätze nur schwer in der Praxis umgesetzt werden können [VOGEL-HEUSER ET AL. 2013]. Als Konsequenz ergibt sich die bereits veranschaulichte Situation (→ 2.1), dass in der Praxis Steuerungsprogramme gar nicht oder nur informell, zum Beispiel durch Text, spezifiziert und direkt implementiert werden.

Der in dieser Arbeit verfolgte Forschungsansatz betrachtet aus diesem Grund das grafische Beschreibungsmittel GRAFCET, welches gemäß internationalem Standard IEC 60848 definiert ist und der grafischen Spezifikation von Steuerungsabläufen dient [IEC 60848 A][#]. Ein systematischer Steuerungsentwurf, der eine automatische Generierung IEC 61131-3 konformen Steuerungscode auf der Grundlage eines Grafcet ermöglicht, beinhaltet vielversprechendes Potential, in der Praxis systematisch eingeführt und durch die Anwender akzeptiert zu werden. Dies liegt darin begründet, dass GRAFCET seit dem Jahr 2005 fester Bestandteil von Zwischen- und Abschlussprüfungen in der beruflichen Ausbildung zukünftiger Steuerungsprogrammierer ist und Einzug in die entsprechenden Lehrpläne gehalten hat. Damit wächst in den kommenden Jahren eine neue Generation von Steuerungsprogrammierern heran, die dieses Beschreibungsmittel in ihrer Ausbildung gelernt haben und somit ohne Einstiegshürde in der praktischen Anwendung nutzen können. Allerdings handelt es sich bei GRAFCET um ein semi-formales grafisches Beschreibungsmittel, so dass formale Methoden, insbesondere zur (teil-) automatischen Generierung, nur dann allgemeingültig angewendet werden können, wenn Struktur und dynamisches Verhalten zuvor mathematisch eindeutig definiert worden sind. Wie in den weiterführenden Kapiteln aufgezeigt werden wird (→ 5), gibt es unterschiedliche Ansätze für die Formalisierung von GRAFCET, welche die Definitionen der IEC 60848 allerdings nur unvollständig umfassen. Ein vollständiges formales Modell existiert bisher nicht. Zur Spezifikation komplexer Steuerungsabläufe im Rahmen eines systematischen Steuerungsentwurfs sollte möglichst der gesamte Sprach-

umfang von GRAFCET nutzbar sein, daher ist ein vollständiges formales Modell zwingend erforderlich.

Ziel des in dieser Arbeit vorgeschlagenen formalen Ansatzes ist es daher, GRAFCET zu verwenden, um Steuerungen zu spezifizieren und eindeutige Transformationsalgorithmen zu entwickeln, mit denen möglichst alle in IEC 60848 definierten Konstrukte genutzt und in IEC 61131-3 konformen Steuerungscode überführt werden können. Hierbei sollen insbesondere die bestehenden Ähnlichkeiten zwischen GRAFCET und SFCs ausgenutzt werden, um die wesentlichen logischen Zusammenhänge bezüglich des dynamischen Verhaltens im Steuerungscode selbst beizubehalten und so die Übersichtlichkeit des Steuerungsprogramms zu gewährleisten. Hierzu werden zunächst die grundlegenden Eigenschaften von SFCs (→ 3) und GRAFCET (→ 4) erläutert. Der Stand der Forschung (→ 5) zeigt auf, welche formalen Methoden bezüglich GRAFCET derzeit verfügbar sind und aus welchen Aspekten sich weiterer Forschungsbedarf ergibt. Der daraus abgeleitete Ansatz zur automatischen Generierung IEC 61131-3 konformen Steuerungscode (→ 8) erfordert zunächst ein formales Modell, welches alle grafischen Elemente von GRAFCET berücksichtigt (→ 6) sowie die Bereitstellung der relevanten Daten in einer implementierungsunabhängigen Notation (→ 7). Am Beispiel softwarebasierter Werkzeuge (→ 9) sowie anhand eines Anwendungsbeispiels (→ 10) wird aufgezeigt, wie der Anwender durch die entwickelte Methode zur automatischen Generierung in einem systematischen Steuerungsentwurf mit GRAFCET gezielt unterstützt werden kann.

3 Sequential Function Charts gemäß IEC 61131-3

3.1 Grundlagen zu Sequential Function Charts

Mit Einzug von SPSen in die industrielle Automatisierungstechnik zu Beginn der 1970er Jahre erfolgte auch die Einführung entsprechender Programmiersprachen zu deren Konfiguration. Eine zunehmende Verbreitung von SPSen und die durch zunächst proprietäre Systeme eingeschränkte Interoperabilität und Austauschbarkeit von Steuerungshardware und -software führten in den 1980er Jahren zu Bestrebungen auf internationaler Ebene, einen einheitlichen Standard für SPSen zu definieren. 1992 wurde erstmals der internationale Standard IEC 61131-3 veröffentlicht, der 1993 auch vom Deutschen Institut für Normung angenommen wurde [DIN EN 61131-3][#] und insgesamt fünf Programmiersprachen für SPSen definiert. Die Definitionen umfassen zwei textuelle, *AnWeisungsListe* (AWL) und *Strukturierter Text* (ST), sowie drei grafische Programmiersprachen, *KontaktPlan* (KOP), die *FunktionsBaustein Sprache* (FBS) und *Sequential Function Charts* (SFC). SFCs sind in Deutschland auch unter dem Begriff *Ablaufsprache* geläufig und dienen im Wesentlichen der grafischen Beschreibung des logischen Ablaufs von SPS-Programmen. Die aktuell gültige Norm aus dem Jahre 2003 [DIN EN 61131-3][#] stellt die zweite Fassung dieses Standards dar und hat sich bis heute im industriellen Umfeld als Programmierstandard weitgehend durchgesetzt. Erst mit der im Jahr 2013 erschienenen dritten Fassung der IEC 61131-3 [IEC 61131-3][#] hält das Konzept der objektorientierten Programmierung als Erweiterung des Funktionsbausteinkonzepts Einzug in die Norm, wie am Beispiel des Programmierwerkzeugs *CODESYS* [CODESYS][@] in [WITSCH & VOGEL-HEUSER 2009] erläutert wird. In Bezug auf SFC sind allerdings keine entscheidenden Änderungen zu verzeichnen. Im Detail handelt es sich bei SFC um eine grafische Programmiersprache zur Implementierung von Steuerungsabläufen in SPSen. SFCs beschreiben den Steuerungsablauf in Form von Schrittketten unter besonderer Berücksichtigung der

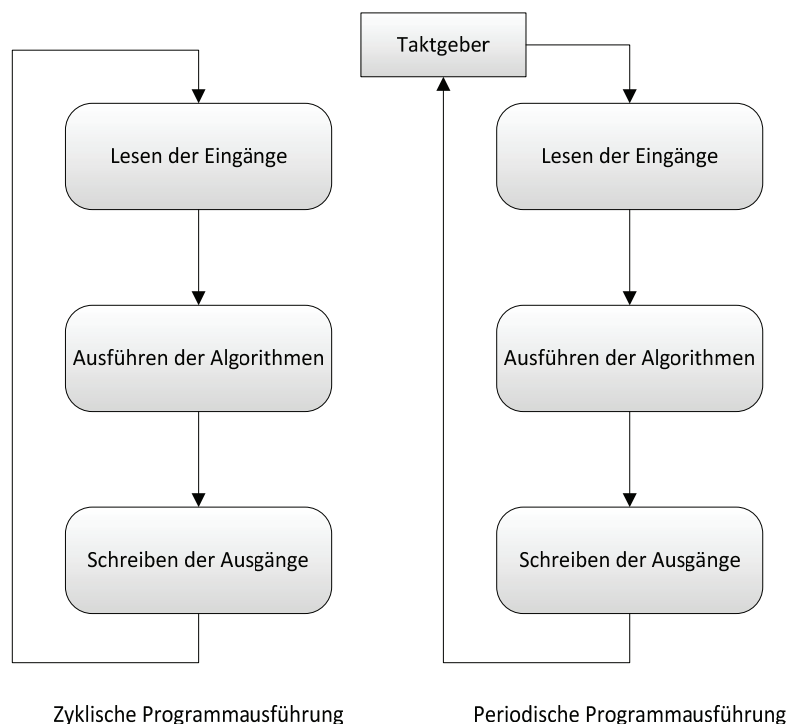


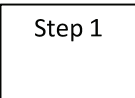
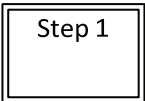

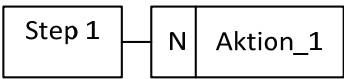
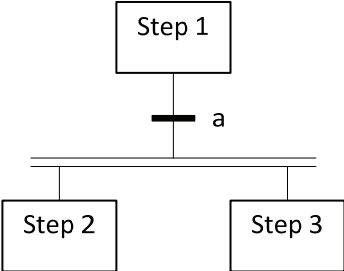
Abbildung 3-1: Zyklische oder periodische Programmausführung in einer SPS, in Anlehnung an [DIN EN 61131-3, Beiblatt 1][#]

technologiespezifischen Anforderungen an eine SPS, wie der typischerweise zyklischen oder periodischen Arbeitsweise (→ Abbildung 3-1) [DIN EN 61131-1][#]. SFCs [DIN EN 61131-3][#] werden nachfolgend in ihren wesentlichen strukturellen und dynamischen Eigenschaften beschrieben sowie Besonderheiten aufgezeigt.

3.2 Struktur und Wirkungsteil

Die Struktur eines SFC besteht aus **Schritten** und **Transitionen**, zwei disjunkten Knotenmengen, die durch gerichtete Verbindungen miteinander verknüpft sind und deren grafische Repräsentation durch Rechtecke (Schritte) und Balken quer zu den Verbindungslinien (Transitionen) festgelegt ist. Jedem Schritt sind zwei **Schrittmerker** zugeordnet, die zum einen den aktiven (=true) oder inaktiven (=false) Zustand des Schrittes in Form einer Booleschen Variablen und die vergangene Zeit seit der letzten Aktivierung des Schrittes in Form einer Variablen von Datentyp TIME (**Timer-Variable**) speichern. Für jeden Schritt eines SFC werden diese Variablen durch die Endungen *.X bzw. *.T angesprochen, wobei * Platzhalter für den jeweiligen Schrittnamen ist. Die Schrittmerker können beispielsweise in Transitionsbedingungen verwendet werden. Für ein SFC-Programm gilt die Einschränkung, dass nur ein Schritt Initialschritt sein kann, der grafisch durch eine doppelte Umrahmung hervorgehoben ist (→ Tabelle 3-1). Dies bedeutet, dass bei Initialisierung allen Schrittvariablen der Wert false bzw. 0 zugewiesen wird und lediglich die Boolesche Variable des Initialschritts den Wert true bzw. 1 besitzt.

Tabelle 3-1: Grafische und textuelle Repräsentation wesentlicher SFC-Elemente

SFC-Element	Grafische Darstellung	Textform
Schritt		STEP Step1: END_STEP
Initialschritt		INITIAL_STEP Step1: END_STEP
Transition		TRANSITION FROM * TO * := a; END_TRANSITION
Aktion		STEP Step1: Aktion_1(N); END_STEP
Parallelverzweigung		TRANSITION FROM Step1 TO (Step2, Step3) := a; END_TRANSITION

Mit jedem Schritt kann eine implementierungsabhängige Anzahl von **Aktionen** durch das grafische Element des Aktionsblocks assoziiert werden. Durch eine Aktion kann entweder eine Boolesche Variable gesetzt oder eine Anweisung aufgerufen werden, die in einer der in IEC 61131-3 definierten Programmiersprachen implementiert wird, SFC eingeschlossen. Eine Anweisung kann somit als eine Art Unterprogramm verstanden werden, welches in einem bestimmten Zustand des Hauptprogramms aufgerufen und ausgeführt wird. Mit Hilfe eines Aktionsblocks wird die Aktion durch ihren Aktionsnamen referenziert und die Art und Weise der Ausführung durch das Aktionsbestimmungszeichen (**Qualifier**) festgelegt. Die Qualifier bestimmen, wie die durch den Aktionsnamen angesprochene Boolesche Variable oder Aktion ausgeführt wird, wie beispielsweise der *S-Qualifier* für die einmalige speichernde Ausführung (→ Anhang B). Insgesamt werden elf Qualifier definiert, deren Bedeutungen Tabelle 45 in [DIN EN 61131-3][#] zu entnehmen sind. Jeder Transition muss definitionsgemäß eine **Transitionsbedingung** zugeordnet werden, die in einer der textuellen Programmiersprachen oder der grafischen Programmiersprachen KOP oder FBS angegeben ist. Transitionsbedingungen stellen Boolesche Übergangsbedingungen dar, die für einen Zustandsübergang notwendigerweise erfüllt sein müssen. Die immer wahre Transitionsbedingung „[...] muss durch das Symbol 1 oder das Schlüsselwort true dargestellt werden“ [DIN EN 61131-3, S.86][#]. Gerichtete Verbindungen werden in SFCs in der Regel durch vertikale Linien gekennzeichnet, die zwischen den Schritten und Transitionen verlaufen. Die Beschreibung alternativer Steuerungsabläufe erfolgt durch rechtwinklige Verzweigungen, parallele Steuerungsabläufe werden durch eine gesonderte Kennzeichnung (Doppelbalken) grafisch hervorgehoben. Zusätzlich zu einer grafischen Repräsentation definiert die IEC 61131-3 für alle Sprachelemente von SFC eine alternative textuelle Repräsentation, so dass ein äquivalenter SFC beispielsweise auch in der Programmiersprache ST erstellt werden kann. Eine Gegenüberstellung der wesentlichen grafischen SFC-Elemente und deren textueller Repräsentation ist in Tabelle 3-1 ersichtlich.

3.3 Dynamisches Verhalten

Das dynamische Verhalten von SFCs wird durch **Ablaufregeln** definiert. Sie geben grundsätzlich an, unter welchen Voraussetzungen ein SFC zu einem bestimmten Zeitpunkt von einem aktiven Zustand in einen möglichen Folgezustand übergeht und wie sich der somit vollzogene Zustandsübergang auf das Ausgabeverhalten des SFC auswirkt. Dabei bestehen Ähnlichkeiten zu den aus dem Bereich der Petrinetze bekannten Schaltregeln. Der ursprüngliche Zustand eines SFC wird als **Initialzustand** bezeichnet und in der grafischen Repräsentation durch die doppelte Umrahmung eines Schrittes hervorgehoben. Bezüglich eines SFCs ist lediglich ein Initialschritt erlaubt, alle anderen Schritte des SFCs sind zum Zeitpunkt der Initialisierung inaktiv. Die Folgezustände des SFCs werden anschließend durch das Schalten von Transitionen erreicht. Eine Transition schaltet genau dann, wenn alle Schritte in ihrem Vorbereich aktiv sind und die zugehörige Transitionsbedingung, welche als Boolesche Bedingung ausgewertet wird, erfüllt ist. Der Vorbereich der Transition besteht aus einer Menge von Schritten, die eine ausgehende Wirkverbindung besitzen, deren Endpunkt die entsprechende Transition ist. Im Vergleich zu Petrinetzen entspricht diese Schaltbedingung einer schwachen Konzessionsregel, da die Schritte im Nachbereich der Transition nicht in Betracht gezogen werden. Schaltet eine Transition, so werden alle Schritte im Vorbereich deaktiviert und alle nachfolgenden Schritte im Nachbereich aktiviert. Verbunden mit dem aktiven

Zustand oder der Zustandsänderung eines Schrittes ist die Ausführung der jeweils assoziierten Aktionen. Somit ist das Ausgabeverhalten eines SFC nicht allein abhängig vom aktuellen Wert der Eingangsvariablen, sondern zusätzlich abhängig von seinem aktuellen Zustand.

Die Schaltvorgänge können allerdings nicht als ideal angenommen werden, da sie diversen technologischen Randbedingungen unterliegen, die in der zyklischen Arbeitsweise einer SPS und ihrem strukturellen Aufbau begründet liegen. Da im Fall von SPSen die Programmabarbeitung von einem einzigen Prozessor gewährleistet wird, schalten Transitionen, die, bezogen auf die logischen Voraussetzungen, gleichzeitig schalten können, nicht exakt gleichzeitig. Lediglich aus der Perspektive eines Programmzyklus besteht die Anforderung, dass alle zu schaltenden Transitionen möglichst innerhalb eines Zyklus schalten. Sie ist wie folgt im Standard IEC 61131-3 definiert:

„Mehrere Transitionen, die gleichzeitig schalten können, müssen gleichzeitig innerhalb der vorgegebenen Zeitanforderungen der jeweiligen SPS-Implementierung und der Prioritätsregeln schalten“ [DIN EN 61131-3, S.99][#].

Eine Prioritätsregel muss daher in der Programmausführung berücksichtigt werden, ihre exakte Umsetzung ist allerdings implementierungsabhängig und kann somit von Engineering-Werkzeug zu Engineering-Werkzeug variieren [DIN EN 61131-3, BEIBLATT 1][#]. In Bezug auf das dynamische Verhalten von SFCs ist es allerdings entscheidend, in welcher Reihenfolge die einzelnen Transitionen und die zugehörigen Transitionsbedingungen überprüft werden und schalten. Auch die Auswahl der in Betracht gezogenen Transitionen kann zu unterschiedlichem Verhalten führen, abhängig davon, ob in einem Programmablaufzyklus nur die freigegebenen oder stets alle Transitionen überprüft werden. In [BAUER 2003] wird dies mit den Begriffen *lock-step*-Semantik und *maximal-progress*-Semantik umschrieben. Basiert der Algorithmus zur Programmausführung beispielsweise auf der „lock-step“-Semantik, so sind transiente Abläufe in einem SFC, in deren Folge einzelne Schritte einer Schrittkette übersprungen werden, nicht möglich, da jeder aktive SFC-Schritt für minimal einen Programmablaufzyklus aktiv ist [PROVOST ET AL. 2010 A]. Ein weiterer entscheidender Aspekt in Bezug auf die Programmausführung ist zudem, ob zuerst die Aktionen eines SFCs und anschließend die Transitionen ausgewertet und ausgeführt werden oder ob dies in umgekehrter Reihenfolge geschieht.

Weitere Besonderheiten bezüglich des dynamischen Verhaltens von SFCs ergeben sich hinsichtlich der SFC-Aktionen. Die Umsetzung und Steuerung der Aktionsausführung gemäß den Vorgaben des Qualifiers erfolgt innerhalb der Laufzeitumgebung eines SFCs durch die Zuordnung einer Instanz eines sogenannten Aktionssteuerungsbausteins:

„Jede Aktion muss mit dem funktionalen Äquivalent einer Instanz des Funktionsbausteins ACTION_CONTROL [...] verknüpft sein.“ [DIN EN 61131-3, S.93][#].

Der ACTION_CONTROL Baustein ist für den Anwender nicht sichtbar, kann sich aber entscheidend auf das dynamische Verhalten eines SFCs auswirken, da der Standard IEC 61131-3 zwei verschiedene Varianten als zulässig definiert. Dass dies zu werkzeugspezifischen Unterschieden bezüglich des dynamischen Verhaltens eines SFCs führen kann, wird beispielsweise in [BAUER ET AL. 2003] und [BAUER ET AL. 2004 B] aufgezeigt. Im Folgenden sollen diese Aspekte allerdings nicht weiter vertieft werden. Insgesamt betrachtet,

definiert die IEC 61131-3 grundlegende Sachverhalte bezüglich des dynamischen Verhaltens eines SFCs, lässt für die exakte Interpretation der dynamischen Zusammenhänge im Rahmen der Implementierung einer Laufzeitumgebung aber einigen Spielraum. Dies ist vor allem dem abstrakten Charakter der Ablaufregeln geschuldet [HUUCK 2003, S.88]. Als Folge ergeben sich herstellerabhängige, werkzeugspezifische Interpretationen für das dynamische Verhalten von SFCs, die beispielsweise in [HELLGREN ET AL. 2005] gegenübergestellt und diskutiert werden.

Aus der Perspektive der Hersteller von SPS-Programmierwerkzeugen müssen die aufgezeigten Definitionslücken in Bezug auf das dynamische Verhalten von SFCs geschlossen und in eindeutiger Weise in einer Laufzeitumgebung implementiert werden. Als Folge ergeben sich herstellerabhängige und werkzeugspezifische Unterschiede, die nicht selten eine Übertragbarkeit von SFCs und damit die Interoperabilität von Engineering-Werkzeugen verhindern. Aus der Perspektive der Forschung besteht großes Interesse daran, SFCs formalen Methoden zugänglich zu machen, um beispielsweise Aussagen über die Steuerbarkeit bestimmter Zustände oder Deadlock-Analysen automatisiert zu generieren. Hierfür müssen jedoch zunächst die zuvor im Ansatz beschriebenen Interpretationsspielräume im Sinne einer eindeutigen Definition eingegrenzt werden. Eine wesentliche Basis hierfür ist die formale Definition der Semantik von SFCs, die beispielsweise und unter Berücksichtigung herstellerepezifischer Aspekte in [BAUER ET AL. 2002] und [BAUER ET AL. 2004 A] vorgeschlagen wird und auf den Ausführungen von [HUUCK 2003] basiert. Ein weiterer Vorschlag für ein zur Rechenzeit optimales Ausführungsmodell wird in [PIEDRAFITA & VILLARROEL 2008] vorgeschlagen, berücksichtigt aber nur eine Teilmenge von SFCs. Weitere Betrachtungen zur formalen Definition und Analyse von SFCs mit Hilfe von zeitbewerteten Petrinetzen oder der synchronen Programmiersprache *Lustre* [LUSTRE][®] werden in [WIGHTKIN ET AL. 2011] und [KABRA ET AL. 2012] beschrieben.

3.4 Möglichkeiten zur hierarchischen Strukturierung

Wie bereits in den Ausführungen zum generellen Aufbau von SFCs (→ 3.2) beschrieben, können Aktionen Anweisungen enthalten, die in einer der fünf IEC 61131-3 Programmiersprachen AWL, ST, KOP, FBS, SFC deklariert sind. Bezüglich der Strukturierung eines SFCs besteht somit die Möglichkeit, SFCs über Aktionen zu verschachteln bzw. ineinander einzubetten. Eine hierarchische Struktur ergibt sich dadurch jedoch nicht, da in IEC 61131-3 nicht eindeutig definiert ist, welche Auswirkungen sich für den SFC ergeben, wenn der eingebettete SFC deaktiviert wird. Abbildung 3-2 zeigt hierzu einen Ausschnitt eines SFCs, dessen momentaner Zustand durch die aktiven Schritte S2 und S12 gegeben ist. Verbunden mit S2 ist ein Aktionsblock mit der Anweisung Aktion_1, über die der eingebettete SFC mit Aktion_1 aufgerufen wird. Führen in einem nachfolgenden Zyklus die Berechnungen zu einem Schalten von Transition T2, so wird Schritt S1 aktiviert und Schritt S2 deaktiviert, wodurch die Aktion Aktion_1 aufgrund des Qualifiers N ebenfalls deaktiviert wird.

Eine wesentliche Definitionslücke der IEC 61131-3 besteht nun darin, dass nicht eindeutig festgelegt ist, ob der zuletzt aktive Zustand des eingebetteten SFC bei Deaktivierung von Aktion_1 für den weiteren Programmablauf gespeichert wird oder nicht. Wie in [BAUER ET AL. 2004 B] aufgezeigt wird, bieten die meisten Werkzeughersteller die Funktionalität an, SFCs ineinander einzubetten. Das dynamische Verhalten der eingebetteten SFCs unter-

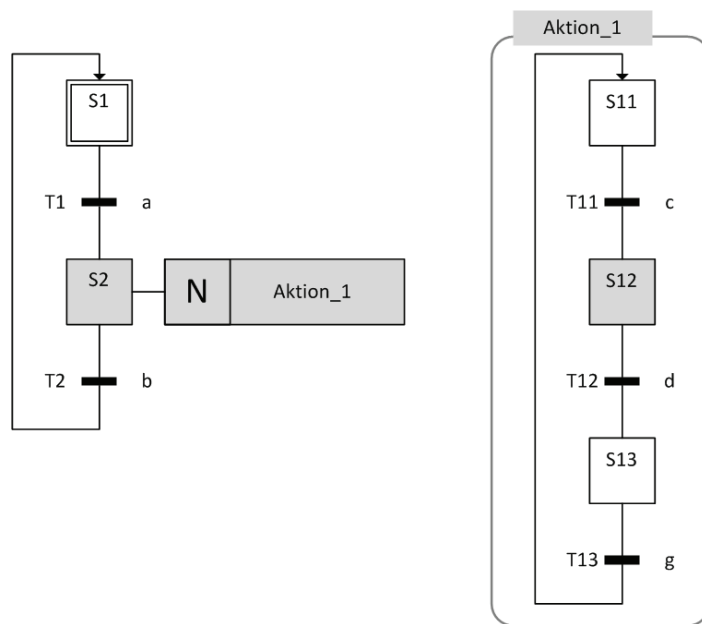


Abbildung 3-2: Einbettung von SFCs durch Aktionen

scheidet sich allerdings von Hersteller zu Hersteller. Dies entspricht nicht den offiziellen Leitlinien in [DIN EN 61131-3, BEIBLATT 1][#], in denen von der Verschachtelung von SFCs abgeraten wird.

Die bisherigen Ausführungen zu SFCs fokussierten auf ein einziges SPS-Programm, welches in der Programmiersprache SFC implementiert ist. Die IEC 61131-3 definiert jedoch nicht nur fünf Programmiersprachen für SPSen, sondern erhebt außerdem Anforderungen an das Softwaremodell allgemein, welches aus ver-

schiedenen *ProgrammOrganisationsEinheiten* (POE) besteht [JOHN & TIEGELKAMP 2000]. Als POE werden Programme, Funktionsbausteine und Funktionen unterschieden. Innerhalb von Programmen können so weitere Programme oder Funktionsbausteine und Funktionen aufgerufen werden, wohingegen aus Funktionsbausteinen nur der Aufruf anderer Funktionsbausteine und Funktionen möglich ist. Innerhalb einer Funktion können lediglich weitere Funktionen, allerdings keine Programme oder Funktionsbausteine aufgerufen werden. Die Ausgaben von Programmen und Funktionsbausteinen sind charakteristischerweise abhängig vom zeitlichen Verlauf der Werte ihrer Eingaben, wohingegen Ausgaben von Funktionen ausschließlich von den aktuellen Werten der Eingaben abhängig sind.

Durch das Softwaremodell und das zugehörige Kommunikationsmodell der IEC 61131-3 ist es möglich, SPS-Projekte zu erstellen, die aus mehreren Programmen, Funktionsbausteinen und Funktionen bestehen [DIN EN 61131-3][#]. Zur Koordination und Steuerung des Ablaufs können den Programmen und Funktionsbausteinen sogenannte Tasks zugewiesen werden, in denen festgelegt wird, wann die jeweilige POE mit welcher Priorität ausgeführt wird. Eine Task kann mehreren POEs, umgekehrt eine POE aber nur einer Task zugeordnet sein. Durch Tasks kann das Echtzeit-/ Antwortverhalten der SPS und ihrer Programmlogik manuell durch eine Prioritätenreihenfolge der einzelnen POEs beeinflusst werden. Sie bieten somit eine weitere Möglichkeit zur Strukturierung von SPS-Programmen. Das Kommunikationsmodell stellt diesbezüglich entsprechende Zugriffspfade zur Verfügung, um nebenläufige Programmausführung und gleichzeitigen Zugriff auf Variablen festzulegen. Die zugehörige Programmiermethode erfordert in der Regel ein hohes Maß an Erfahrung.

Die IEC 61131-3 legt zwei grundsätzliche Möglichkeiten zur Strukturierung von SPS-Programmen fest. In Bezug auf SFC ist entscheidend, dass durch die Verschachtelung über Aktionen Programme entstehen können, die gemäß den Definitionen der Norm in ihrem Ablauf nicht eindeutig definiert und somit herstellerabhängig sind. Weiterhin können dadurch, in Kombination mit einer fehlenden Methode zur systematischen Erstellung von SFCs,

leicht unübersichtliche SFCs resultieren, deren Verhalten für den Anwender nur schwer nachvollziehbar ist. Aus diesem Grund untersuchen [FREY & LITZ 1997] ausgehend von dem Begriff der *Transparenz*, wie SFCs verständlich und nachvollziehbar erstellt werden sollten. Die Empfehlungen zu einer transparenten Erstellung von SFCs und das Wissen über Definitionslücken und kritische Konstrukte der IEC 61131-3 führen beispielsweise in [BAUER 2003] zur Definition eines *implementierungsunabhängigen SFC*. Für den weiteren Verlauf dieser Arbeit soll das Hauptaugenmerk auf die Programmiersprache SFC, gemäß den Definitionen der IEC 61131-3 und unabhängig von werkzeugspezifischen Implementierungen, gelegt werden, ohne eine bestimmte SPS-Projektstruktur und die Zuordnung von Tasks in Betracht zu ziehen.

3.5 Zeitabhängige Bedingungen

Zeitabhängige Bedingungen in SFC können durch die jedem SFC-Schritt zugeordnete Timer-Variable oder die zeitverzögerte bzw. zeitbegrenzte Ausführung von Aktionen festgelegt werden. Die Timer-Variable eines Schrittes bietet die Möglichkeit, den aktiven Zustand eines Schrittes zeitlich zu begrenzen, wenn sie beispielsweise in einer dem Schritt nachfolgenden Transitionsbedingung verwendet wird, wie im Falle des Warteschritts S6 in Abbildung 3-3 gezeigt. Hier schaltet die Transition im Nachbereich fünf Sekunden nach der Aktivierung von

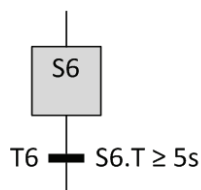


Abbildung 3-3: Zeitabhängige Bedingung durch Nutzung einer Timer-Variablen

Schritt S6. S6 wird dann deaktiviert. Werden entsprechende Konstrukte als zeitabhängige Bedingungen verwendet, so ist anzumerken, dass es sich bei den Schrittvariablen jeweils um lokale Variablen der jeweiligen POE handelt. Eine POE-übergreifende Nutzung von Schrittvariablen ist gemäß IEC 61131-3 nicht vorgesehen.

Für SFC-Aktionen stehen insgesamt fünf Qualifier zur Verfügung, die eine zeitverzögerte oder zeitbegrenzte Ausführung gewährleisten. Die zeitbegrenzte Aktion wird grundsätzlich durch den L-Qualifier, die zeitverzögerte Aktion durch den D-Qualifier bestimmt. Ein notwendiges Kriterium für die Ausführung dieser Aktionen ist der aktive Zustand des assoziierten Schrittes. Darüber hinaus erlauben die Qualifier SD, DS und SL eine speichernde zeitverzögerte oder zeitbegrenzte Aktionsausführung. Eine detaillierte Beschreibung des Verhaltens von SFC-Aktionen unter Berücksichtigung der Qualifier soll an dieser Stelle nicht erfolgen. Es sei diesbezüglich auf [DIN EN 61131-3, BEIBLATT 1][#] verwiesen.

Aufgrund der Fokussierung auf *implementierungsunabhängige SFCs* werden im Rahmen dieser Arbeit ausschließlich die Qualifier *N, S, R, L, D* in Betracht gezogen (→ Anhang B).

3.6 Werkzeugunterstützung

Hinsichtlich der Werkzeugunterstützung gibt es in Bezug auf die IEC 61131-3 und insbesondere SFC eine Vielzahl softwarebasierter Werkzeuge, die für die Programmierung von SPSen im Engineering automatisierter technischer Systeme eingesetzt werden. So bieten die meisten Hersteller Speicherprogrammierbarer Steuerungen ausgereifte, in der praktischen Anwendung etablierte Werkzeuge mit hoher Funktionsintegration in Ergänzung zu ihrer Automatisierungs-Hardware an. Beispiele für weit verbreitete Programmierwerkzeuge sind *STEP 7* von Siemens [STEP 7][®], *CODESYS* von 3s-Software [CODESYS][®], *Multiprog* von KW-Software [MULTIPROG][®] oder *Control Build* von Dassault Systèmes [CONTROL BUILD][®]. *CODESYS* ist beispielsweise integraler Bestandteil des SPS-Programmierwerkzeugs *Control Builder* von ABB [CONTROL BUILDER][®] oder *TwinCAT* von Beckhoff [TWINCAT][®]. Für Steuerungen von Phoenix Contact existiert das Programmierwerkzeug *PC WORX* [PCWORX][®], welches auf *Multiprog* basiert. Daneben existieren weitere SPS-Programmierwerkzeuge, wie beispielsweise *Logi.CAD* der Firma logi.cals [LOGI.CALS][®].

Die IEC 61131-3 wird als allgemein akzeptierter Industriestandard in unterschiedlichem Umfang unterstützt. Insbesondere in Bezug auf SFCs können sich herstellerabhängig unterschiedliche Implementierungen des Programmablaufs ergeben [BAUER 2003]. Inwieweit die einzelnen Werkzeuge die Vorgaben der IEC 61131-3 erfüllen, wird beispielsweise durch die *PLCopen*-Organisation in Form eines Zertifizierungsverfahrens untersucht [PLCOPEN][®]. *PLCopen* ist eine unabhängige Organisation, die bestrebt ist, einheitliche Standards im Engineering Speicherprogrammierbarer Steuerungen zu entwickeln und voranzutreiben. Die Zertifizierung ist allerdings nicht bindend für die Hersteller, so dass nach wie vor herstellerabhängige und werkzeugspezifische Unterschiede für die Implementierung der Definitionen der IEC 61131-3 bestehen. Um dennoch, trotz der angestrebten Alleinstellungsmerkmale von Seiten der Hersteller, einen vereinfachten Datenaustausch für Steuerungsalgorithmen gemäß IEC 61131-3 zu etablieren, wurde im Juni 2005 die erste Version eines XML-basierten Datenaustauschformates durch die *PLCopen*-Organisation veröffentlicht. Das sogenannte *PLCopenXML-Schema* wurde im Einvernehmen zwischen Anwendern und Herstellern IEC 61131-3 konformer Programmierwerkzeuge beschlossen und ist seit Dezember 2008 in der zweiten, überarbeiteten Version verfügbar [PLCOPEN 2009][#]. Eine umfassende Werkzeugunterstützung bleibt bisher, bis auf wenige Ausnahmen, jedoch aus. So ergeben sich herstellerabhängig Unterschiede, welche Programmiersprachen der IEC 61131-3 über das *PLCopenXML*-Format in das jeweilige Werkzeug importiert oder exportiert werden können (→ Tabelle 3-2). So bietet *STEP 7* keine Schnittstelle zum *PLCopenXML*-Format an. *CODESYS* unterstützt den Import und Export von *PLCopenXML*-Dateien, allerdings nur für die Programmiersprache ST. SFCs, die in textueller Form in ST beschrieben sind, können in der aktuellen *CODESYS*-Version 3.5 zwar importiert, aber nicht kompiliert werden. *Logi.CAD* unterstützt das *PLCopenXML*-Format bezüglich des Im- und Exports lediglich für die Programmiersprache FBS. Nur *Multiprog* und *Control Build* bieten eine *PLCopenXML*-Schnittstelle für den Im- und Export von grafischen SFCs an. Im Falle von *Control Build* werden wesentliche SFC-Daten als werkzeugspezifisch deklariert, so dass ein werkzeugübergreifender Datenaustausch nur schwer möglich ist. Eine Unterstützung von textuell beschriebenen SFCs ist in beiden Fällen allerdings nicht vorhanden.

Tabelle 3-2: Werkzeugunterstützung für Import und Export von SFCs im *PLCopenXML*-Format

Werkzeugfunktionen bezüglich <i>PLCopenXML</i>	SPS-Programmierwerkzeug				
	<i>STEP 7</i>	<i>CODESYS</i>	<i>Multiprog</i>	<i>Control Build</i>	<i>Logi.CAD</i>
Importfunktion verfügbar	-	+	+	+	+
Exportfunktion verfügbar	-	+	+	+	+
Import von SFCs (grafisch) möglich	-	-	○ ¹	○ ²	- ³
Export von SFCs (grafisch) möglich	-	-	○ ¹	○ ²	- ³
Import von SFCs (textuell, ST) möglich	-	○	-	-	-
Export von SFCs (textuell, ST) möglich	-	+	-	-	-

Legende: +: erfüllt/vorhanden/möglich
 ○: teilweise erfüllt/teilweise vorhanden/teilweise möglich
 -: nicht erfüllt/nicht vorhanden/nicht möglich

¹ gilt für *PLCopenXML*-Schema Version 1.01

² kein werkzeugübergreifender Datenaustausch möglich

³ Unterstützung nur für FBS

4 GRAFCET gemäß IEC 60848

4.1 Hintergründe

4.1.1 Der Weg zur internationalen Norm

GRAF CET ist ein Akronym für *GRA*ph *Fon*ctionnel de *Com*mande *Et*ape *Tr*ansition [IEC 60848 A][#] und bedeutet wörtlich übersetzt *Graph zur funktionalen Beschreibung der Steuerung in Form von Schritten und Transitionen*. In der heute auch in Deutschland angenommenen gültigen Norm DIN EN 60848 aus dem Jahre 2002 wird GRAFCET als „Spezifikationsprache für Funktionspläne der Ablaufsteuerung“ [DIN EN 60848][#] bezeichnet und somit der Anwendungsbereich des Beschreibungsmittels explizit hervorgehoben. So steht die Spezifikation, also „*die funktionale Beschreibung des Verhaltens des Ablaufteils eines Steuerungssystems*“ [DIN EN 60848, S.5][#], im Fokus von GRAFCET. Hierzu werden grafische Elemente definiert, die eine Modellierung des Verhaltens einer Steuerung ermöglichen.

Ursprünglich stammt GRAFCET aus Frankreich. Die Anfänge reichen zurück an den Anfang der 1970er Jahre, als SPSen erstmals Einzug in die Automatisierung industrieller Systeme und Anlagen hielten [AUER 1991]. Durch die Einführung von SPSen wurde die Programmlogik der Steuerung nunmehr in Form von Software und nicht, wie bisher üblich, in einer Hardware-Struktur als verbindungsprogrammierte Relais-Schaltung implementiert [SEITZ 2003]. Diese neue technische Realisierungsart einer Steuerung machte gleichzeitig ein Umdenken in Bezug auf den Entwurf und die Programmierung solcher Automatisierungssysteme erforderlich. Um die Akzeptanz der Anwender zu erhöhen, boten die SPS-Hersteller von Beginn an neben assemblernahen auch grafische Programmiersprachen an, mit denen die gewohnten Relais-Schaltungen und Stromlaufpläne auf dem Computerbildschirm nachgebildet werden konnten. Die hardwareorientierte Sichtweise auf das Automatisierungssystem setzte sich somit in Form der auch heute noch im IEC-Standard definierten Programmiersprachen FUP und KOP fort [LITZ & FREY 1999]. Dennoch erfordert diese Art der Programmerstellung viel Erfahrung, ist wenig intuitiv und für spätere Dokumentationszwecke eher ungeeignet, in der praktischen Anwendung allerdings bis heute weit verbreitet [LUCAS & TILBURY 2004].

Zeitgleich mit der Einführung Speicherprogrammierbarer Steuerungen beschäftigte sich in Frankreich die 1968 aus verschiedenen Vorgängerorganisationen hervorgegangene Gesellschaft für Kybernetik, Wirtschaft und Technik *Association Francaise pour la Cybernétique Economique et Technique* (AFCET) [Hoffsaes 1990] mit der Entwicklung einer methodischen Vorgehensweise im Steuerungsentwurf. Jeweils zur Hälfte zusammengesetzt aus Vertretern der Industrie und der Forschung, war die Zielsetzung der Arbeitsgruppe *Systèmes Logiques*, Anforderungen an ein einheitliches und umfassendes Beschreibungsmittel für Automatisierungssysteme herauszuarbeiten und zu dokumentieren. Als Ergebnis legte die AFCET Mitte der 1970er Jahre einen Abschlussbericht vor, der GRAFCET in seinen Grundzügen eindeutig beschreibt. Vor allem die seit 1962 bekannten und auf Carl Adam Petri zurückzuführenden Petrinetze [PETRI 1962] bildeten hierbei die wesentliche konzeptionelle Grundlage. Nur wenige Jahre später, im Oktober 1977, gründete sich eine Arbeitsgruppe im nationalen industriellen Interessenverband *Agence pour le Développement de la Productique*

Appliquée à l'industrie (ADEPA). Aufbauend auf dem Ergebnisbericht der AFCET-Kommission sollte zunächst eine französische und später eine internationale Norm für GRAFCET erarbeitet und die Akzeptanz von GRAFCET zur Spezifikation von Steuerungsabläufen auf nationaler Ebene gesteigert werden. Zur Nutzung von Synergieeffekten wurde auch eine Zusammenarbeit von ADEPA und AFCET beschlossen [GREPA 1985].

Die Aktivitäten von ADEPA und AFCET mündeten bereits 1978 in eine erste Erprobungsphase der Anwendung von GRAFCET auf Ebene der schulischen Ausbildung in technischen Gymnasien in Frankreich und im Jahre 1982 in die nationale französische Norm NF C 03-190 [NF C 03-190][#]. Ergänzend zum Steuerungsentwurf mit GRAFCET entstand durch die Arbeiten der vorgenannten Arbeitsgruppen auch der sogenannte *Guide d'Étude des Modes de Marches et d'Arrêts* (GEMMA), eine methodische Vorgehensweise unter Berücksichtigung der Betriebszustände eines technischen Systems [ADEPA 1981][#], [CLOUTIER & PAQUES 1988]. Weiterführend zu den Arbeiten von AFCET und ADEPA schlossen sich in den Anfangsjahren von GRAFCET Forscher und Anwender in der sogenannten *Groupe GRAFCET* zusammen, um die praktische Anwendung im Lebenszyklus automatisierter technischer Systeme sowie die theoretischen Grundlagen von GRAFCET zu erforschen und zu etablieren [LURPA][@]. Die *Groupe GRAFCET* besteht bis heute und hat zahlreiche bedeutende Forschungsergebnisse hinsichtlich GRAFCET hervorgebracht.

Im Jahre 1988 wurde GRAFCET erstmals international genormt und als Spezifikations-sprache für Ablaufsteuerungen definiert. Unter dem Titel *Preparation of function charts for control systems* führte die *International Electrotechnical Commission* (IEC) die Norm IEC 848 ein [IEC 848 A][#], welche später in die deutsche Norm DIN 40719-6 „Regeln für Funktionspläne“ [DIN 40719-6][#] einfluss. Bereits 1992/93 wurde eine neue Version der IEC 848 gültig [IEC 848 B][#]. Sie enthielt einige Modifikationen bezüglich der Notation und auch neue Sprachmittel zur Strukturierung von Graficets, wie beispielsweise Makroschritte (→ 4.3). Einen guten Überblick über diese Modifikation sowie die Diskussion weiterführender Sprachkonstrukte, die teilweise in die aktuell gültige Norm eingeflossen sind, geben beispielsweise [DAVID & ALLA 1992] und [GUILLEMAUD & GUÉGUEN 1999].

Aufgrund einer Neunummerierung aller IEC Normen im Jahre 1996 führte die Norm 848 von nun an die Nummer 60848. Im üblichen Revisionszyklus wurde 2002 die zweite Version der IEC 60848 [IEC 60848 A] verabschiedet, durch das Europäische Komitee für Elektrotechnische Normung (CENELEC) als Europäische Norm anerkannt und als Deutsche Norm umgesetzt [DIN EN 60848][#]. Die Norm führte von nun an GRAFCET explizit im Titel, wie der deutsche Normtitel „GRAFCET, Spezifikations-sprache für Funktionspläne der Ablaufsteuerung“ zeigt. Die IEC 60848 wurde ohne Abänderung als deutsche Norm DIN EN 60848 übernommen und ersetzte mit Ablauf des 01.04.2005 die bisher gültige Norm für Funktionspläne DIN 40719-6 vollständig. In einem weiteren Revisionszyklus wurde die IEC 60848 nochmals leicht überarbeitet und liegt seit dem 27.02.2013 als dritte Auflage [IEC 60848 B][#] des Standards vor. Die historische Entwicklung des Beschreibungsmittels GRAFCET, von den ersten Gremienarbeiten bis hin zur internationalen Normung, ist in einer tabellarischen Übersicht in Anhang A dieser Arbeit beigefügt. Eine ausführliche Gegenüberstellung des Standards von 1988 und der Version von 2002 findet sich beispielsweise in [GUÉGUEN & BOUTEILLE 2001].

4.1.2 Anwendungsbereich von GRAFCET

Grundsätzlich definiert die DIN EN 60848 GRAFCET als „[...] eine grafische Entwurfssprache für die funktionale Beschreibung des Verhaltens des Ablaufteils eines Steuerungssystems“ [DIN EN 60848, S.5][#]. Ein Grafcet ist demzufolge ein grafisches Modell für das gewünschte Verhalten einer Steuerung, welches dem Anwender veranschaulicht, was die Steuerung zu welchem Zeitpunkt tun soll. Der Grafcet berücksichtigt die funktionalen Anforderungen an den Steuerungsablauf und abstrahiert von dessen technologischer Realisierung, wie beispielsweise einem SPS-Programm. Dieser Abstraktionsgrad ermöglicht es dem Anwender, das Verhalten der Steuerung zunächst allgemeingültig festzulegen, zu testen und anzupassen, um es anschließend in eine Implementierung zu überführen. Aufgrund einer in der heutigen Praxis fehlenden eindeutigen Abgrenzung des Steuerungsentwurfs von der Implementierung der Steuerung (→ 2.1) werden die Begriffe GRAFCET und SFC häufig synonym verwendet [ROUSSEL ET AL. 1999]. Im Gegensatz dazu wird im Rahmen dieser Arbeit unter GRAFCET die Spezifikationsprache gemäß IEC 60848 und unter SFC die Programmiersprache gemäß IEC 61131-3 verstanden.

Die IEC 60848 definiert grafische Symbole (→ Anhang C), die in einer eindeutigen Struktur (**Syntax**) angeordnet werden müssen, um einen gültigen Grafcet zu erstellen. Jedes grafische Symbol besitzt eine festgelegte Bedeutung (**Semantik**) sowie charakteristische Eigenschaften, die in Kombination mit der Struktur ein bestimmtes Verhalten der Steuerung spezifizieren. Das Verhalten der Steuerung wird als eine stets wechselnde Abfolge von Zuständen und Zustandsübergängen angesehen, wodurch eine zeitliche Reihenfolge entsteht. Insofern wird der Parameter Zeit in einem Grafcet nicht als kontinuierliche Größe im Modell berücksichtigt, sondern nur zu diskreten Zeitpunkten, wie auch die Eingaben eines Grafcet nur zu diskreten Zeitpunkten, sogenannten Ereignissen, berücksichtigt werden. Auch die Ausgaben eines

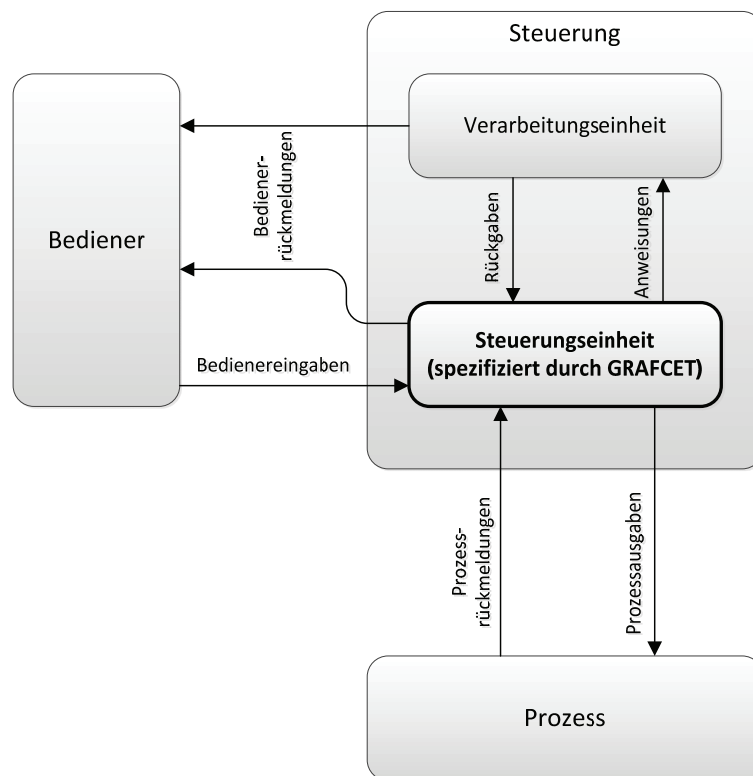


Abbildung 4-1: Schematische Darstellung einer Steuerung, in Anlehnung an [DAVID & ALLA 2010]

Graf cet werden als diskrete Größen gehandhabt.

Insgesamt betrachtet, modelliert GRAFCET also das logische Verhalten einer Steuerung in Gestalt eines ereignisdiskreten Systems [DAVID 1995]. Eine genauere Betrachtung der Systemgrenzen von GRAFCET wird beispielsweise in [DAVID & ALLA 2010] aufgezeigt und ergibt näheren Aufschluss darüber, was unter dem Ablaufteil des Steuerungssystems zu verstehen ist und wie GRAFCET im Rahmen dieser Arbeit verstanden wird. Abbildung 4-1 zeigt hierzu den stark vereinfachten und schematischen Aufbau einer Steuerung, die einerseits über Ein- und Ausgaben mit dem Prozess bzw. dem zu automatisierenden technischen System in Wechselwirkung steht (**Prozessausgaben und -rückmeldungen**). Andererseits besteht eine Wechselwirkung mit der Bedienerschnittstelle über **Bedienereingaben und -rückmeldungen**. Die Steuerung selbst unterteilt sich in die **Steuerungseinheit** und die **Verarbeitungseinheit**. In der Verarbeitungseinheit werden die Werte von solchen Variablen abgespeichert und verarbeitet, die zur Speicherung des internen Zustandes der Steuerung notwendig sind, wie beispielsweise Zähler. Die Verarbeitungseinheit steht als Empfänger von Anweisungen und Sender von Rückgaben in Wechselwirkung mit der Steuerungseinheit und wird als reaktive Komponente der Steuerung betrachtet. In der Steuerungseinheit findet die Berechnung der Prozessausgaben und Bedienerückmeldungen der Steuerung statt. Dies geschieht aktiv gemäß den Vorgaben zum logischen Ablauf der Steuerung und unter Berücksichtigung der aktuellen Bedienereingaben und Prozessrückmeldungen. Das Verhalten der Steuerungseinheit über die Systemgrenze hinweg wird charakterisiert durch das Ausgabeverhalten, welches von dem Verlauf der Werte der Eingaben abhängig ist.

Der Ablaufteil einer Steuerung ist gemäß den vorangegangenen Ausführungen dem Steuerungsteil des in Abbildung 4-1 veranschaulichten Steuerungsmodells zuzuordnen und wird in seinen logischen Zusammenhängen durch GRAFCET spezifiziert. GRAFCET umfasst aufgrund seines Abstraktionsniveaus sowohl die Hardware als auch die Software des Steuerungsteils, ohne auf technologiespezifische Eigenschaften einzugehen.

4.2 Grundlegende Eigenschaften

4.2.1 Struktur

Die Struktur eines Graf cet besteht aus zwei disjunkten Knotenmengen, so genannten **Schritten** und **Transitionen**, die durch **Wirkverbindungen** abwechselnd miteinander verbunden sind. Schritte werden grafisch durch Quadrate, Transitionen durch waagerechte Balken und Wirkverbindungen durch direkte Verbindungen zwischen Schritt und Transition dargestellt [DIN EN 60848][#]. Die Richtung einer Wirkverbindung ist per Konvention von oben nach unten festgelegt und wird nur dann durch eine Pfeilspitze gekennzeichnet, wenn die Richtung von dieser Konvention abweicht.

Alle Transitionen, die unmittelbar durch eine Wirkverbindung mit dem betroffenen Schritt als Ziel verbunden sind, zählen zum **Vorbereich des Schrittes**. Diejenigen Transitionen, welche unmittelbar durch eine Wirkverbindung mit dem betroffenen Schritt als Ausgangspunkt verbunden sind, gehören zum **Nachbereich des Schrittes**. Entsprechendes gilt für den Vor- und Nachbereich von Transitionen. Ein Schritt ist in seinem Vor- und Nachbereich stets mit einer oder mehreren Transitionen, eine Transition in ihrem Vor- und Nachbereich stets mit einem oder mehreren Schritten verbunden. Sollte eine Transition in ihrem Vor- und/oder

Nachbereich mit mehreren Schritten verbunden sein, so werden die entsprechenden einzelnen Wirkverbindungen grafisch durch einen Doppelbalken ersetzt, wie das Beispiel eines Grafcet in Abbildung 4-2 (links) zeigt. Der Doppelbalken ist das Symbol für eine parallele Verzweigung und wird auch als **Synchronisierungssymbol** bezeichnet [DIN EN 60848][#]. Ist ein Schritt in seinem Vor- und/oder Nachbereich mit mehreren Transitionen verbunden, so verlaufen die einzelnen Wirkverbindungen, wie in Abbildung 4-2 (rechts) dargestellt, in einem teilweise gemeinsamen Fluss zum Schritt hin bzw. vom Schritt weg. Diese Art der Darstellung kennzeichnet alternative Verzweigungen [DIN EN 60848][#].

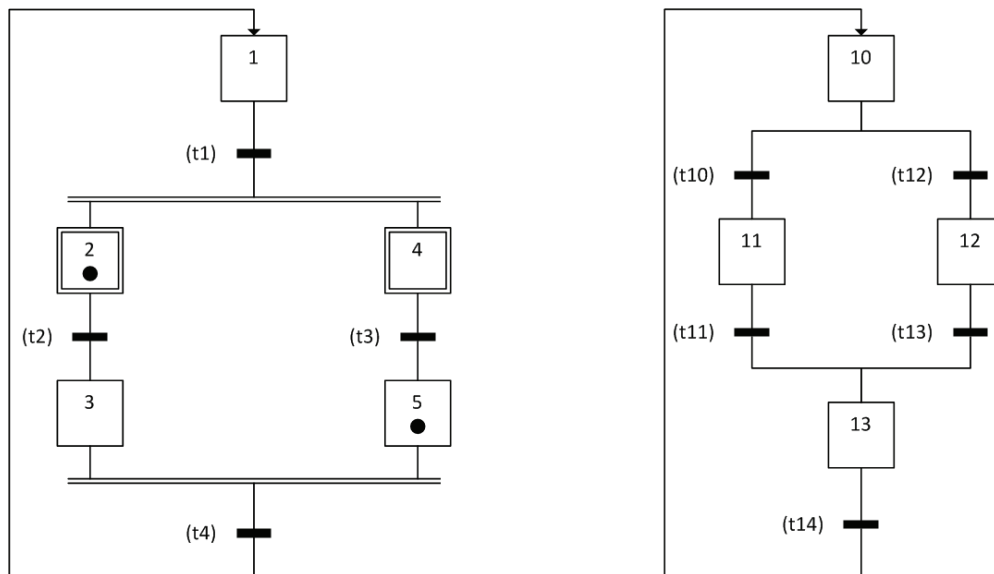


Abbildung 4-2: Struktur von GRAFCET mit Parallel- (links) und Alternativverzweigung (rechts)

Jeder Schritt und jede Transition wird mit einem Namen versehen, um eine eindeutige Identifikation der einzelnen Elemente zu gewährleisten. Der Name des Schrittes wird innerhalb des Schrittsymbols angezeigt, die Transitionennamen werden links von den Transitionen dokumentiert. Ein Schritt besitzt zwei diskrete Zustände und kann entweder aktiv oder inaktiv sein. Jedem Schritt ist eine Boolesche Variable zugeordnet, die den Zustand des Schrittes repräsentiert und deren Wert den aktuellen Zustand des Schrittes zurückgibt (*true* = aktiv, *false* = inaktiv). Die Menge der zu einem bestimmten Zeitpunkt aktiven Schritte wird als **Situation** des Grafcet bezeichnet und grafisch durch die Darstellung einer Marke innerhalb des Schrittsymbols hervorgehoben, wie anhand der Schritte 2 und 5 in Abbildung 4-2 ersichtlich. Die Situation bei Initialisierung eines Grafcet ist die Menge derjenigen Schritte, die durch eine doppelte Umrahmung und somit als Initialschritt gekennzeichnet sind. Zu beachten ist, dass ein Grafcet nicht *exakt* einen, sondern *mindestens* einen Initialschritt besitzt, ähnlich zur Anfangsmarkierung in Petrinetzen [DAVID 1995].

4.2.2 Wirkungsteil

Allein die Struktur eines Grafcet (→ 4.2.1) vermag noch nicht das logische Verhalten einer Steuerung vollständig zu beschreiben. In der in Abbildung 4-2 gezeigten Form ist der Grafcet zunächst autonom, da er keine Ein- und Ausgaben besitzt. Zudem fehlt es an Regeln, die festlegen, unter welchen Bedingungen ein Grafcet von der momentanen Situation (beispielsweise der Initialsituation) in eine nachfolgende Situation übergeht (→ 4.2.3). Der Bezug zwischen Struktur, Eingaben und Ausgaben wird in GRAFCET durch

sogenannte **Transitionsbedingungen** und **Aktionen** (\rightarrow Anhang C) festgelegt, die den **Wirkungsteil** des Grafset bilden [DIN EN 60848][#].

Jeder Transition ist eine Transitionsbedingung zugeordnet, die innerhalb des Grafset rechts von der jeweiligen Transition platziert wird (\rightarrow Abbildung 4-3). Sie stellt, insgesamt betrachtet, einen logischen Ausdruck dar, der entweder erfüllt ($= true$) oder nicht erfüllt ($= false$) ist und aus einer grundsätzlich unbegrenzten Menge logischer Operanden und Operatoren besteht. Die Ausdrücke *true* und *false* können auch direkt als Transitionsbedingung verwendet werden (Nr. 1, \rightarrow Anhang C), dürfen dann aber nicht in Kombination mit einem weiteren logischen Ausdruck aufgeführt werden. Zu den gültigen logischen Operanden in Transitionsbedingungen zählen Eingaben des Grafset, die durch entsprechende Eingangsvariablen oder Aussagen repräsentiert werden. Als Eingangsvariablen gelten sowohl Prozessrückmeldungen und Bedieneringaben der Steuerung als auch Rückgaben der Verarbeitungseinheit in Form von internen Variablen, wie beispielsweise Zähler und Schrittvariablen des Grafset. Unter einer Aussage fasst die IEC 60848 zum einen informelle Angaben, wie beispielsweise Temperatur OK (Nr. 4, \rightarrow Anhang C), und zum anderen Relationen zusammen, die Eingangsvariablen mit einem numerischen Wert in Beziehung setzen, wie beispielsweise $T \geq 30^\circ\text{C}$. Als logische Operatoren sind für GRAFCET die Booleschen Operatoren UND (\wedge), ODER (\vee), Negation (\bar{x}), die steigende (\uparrow) und die fallende Flanke (\downarrow) vorgesehen, wobei x eine beliebige Boolesche Variable darstellt. Durch die Booleschen Operatoren werden sogenannte Bedingungen definiert (Nr. 2, \rightarrow Anhang C), deren logischer Zustand (*true* oder *false*) von den Zuständen der jeweiligen Eingaben abhängig ist. Im Gegensatz dazu werden durch Verwendung der Flanken-Operatoren sogenannte Ereignisse definiert (Nr. 3, \rightarrow Anhang C), deren logischer Zustand von einem Zustandsübergang der jeweiligen Eingaben abhängig ist [DAVID & ALLA 2010]. So ist beispielsweise der logische Zustand des Ereignisses ($\uparrow a$) nur dann *true*, wenn der logische Zustand der Variablen a von *false* nach *true* wechselt.

Eine Transitionsbedingung kann sowohl Bedingungen als auch ein Ereignis enthalten. Aus diesem Grund können die Transitionsbedingungen unter alleiniger Anwendung der Booleschen Algebra mitunter nicht vollständig ausgewertet werden, so dass für GRAFCET diesbezüglich eine Erweiterung notwendig ist. Bereits in [DAVID & ALLA 1992] wurde die als *Algebra of Events* bezeichnete erweiterte Boolesche Algebra als eine wesentliche Basis für das Verständnis des dynamischen Verhaltens eines Grafset postuliert. Die Annahme, dass beispielsweise zwei voneinander unabhängige Ereignisse nicht zum exakt gleichen Zeitpunkt eintreten können, und weitere wesentlichen Aspekte sind in [DAVID 1995] zusammengefasst. Weitere ähnliche Betrachtungen zur Formalisierung von Ereignissen mit Bezug zu GRAFCET wurden unter anderem in [LESAGE ET AL. 1996] veröffentlicht.

Neben den Transitionsbedingungen bilden die Aktionen eine weitere bedeutende Schnittstelle zur Umgebung eines Grafset. Sie werden ebenfalls dem Wirkungsteil zugerechnet. Insgesamt betrachtet repräsentieren Aktionen Ausgaben, die ein Grafset in Abhängigkeit seiner momentanen Situation über die Systemschnittstellen kommuniziert. In Anlehnung an Abbildung 4-1 können Aktionen Prozessausgaben, Bedienerückmeldungen oder Anweisungen an die Verarbeitungseinheit sein. Jede Aktion wird grafisch durch ein Rechteck dargestellt, rechts neben dem jeweiligen Schritt platziert und diesem durch eine waagerechte Verbindung assoziiert. Die waagerechte Verbindung zeigt an, dass eine logische Verknüpfung

zwischen dem Zustand des Schrittes und der Aktion besteht, eine Aktionsausführung beispielsweise nur dann erfolgt, wenn der zugeordnete Schritt aktiv ist. Den Definitionen in [DIN EN 60848][#] zufolge ist auch die Möglichkeit gegeben, gespeichert wirkende Aktionen mit Transitionen zu verknüpfen. Es handelt sich um sogenannte **Aktionen bei Auslösung**, die auf [GUILLEMAUD & GUÉGUEN 1999] zurückzuführen sind. Einerseits sind die Ausführungen der Norm in diesem Aspekt unzureichend, da eine Abgrenzung zu gespeichert wirkenden Aktionen nicht deutlich wird. Andererseits verstößt eine Aktion bei Auslösung gegen ein Prinzip von GRAFCET, nach dem Eingaben stets den Transitionen und Ausgaben den Schritten zugeordnet werden, wie in [DAVID & ALLA 2010] ausgeführt. Insofern werden Aktionen bei Auslösung im Rahmen dieser Arbeit nicht berücksichtigt.

Für GRAFCET werden zwei Aktionsarten unterschieden, die eine grundsätzlich unterschiedliche Semantik aufweisen, *kontinuierlich wirkende Aktionen* und *gespeichert wirkende Aktionen*. Während *kontinuierlich wirkende Aktionen* in ihrer Ausführungsdauer an eine Bedingung geknüpft sind (Nr. 7-9, → Anhang C), welche beispielsweise der aktive Zustand des ihnen assoziierten Schrittes sein kann, ist die Ausführung einer *gespeichert wirkenden Aktion* an ein Ereignis geknüpft. Ein solches Ereignis kann durch die Aktivierung ($\uparrow X_n$) oder die Deaktivierung ($\downarrow X_n$) des assoziierten Schrittes n (Nr. 10-11, → Anhang C), oder durch ein anderes Ereignis (Nr. 12, → Anhang C), spezifiziert sein. Im letzten Fall wird die *gespeichert wirkende Aktion* nur dann ausgeführt, wenn der assoziierte Schritt aktiv ist und das Ereignis eintritt. Die in Abbildung 4-3 abgebildete *kontinuierlich wirkende Aktion* action3

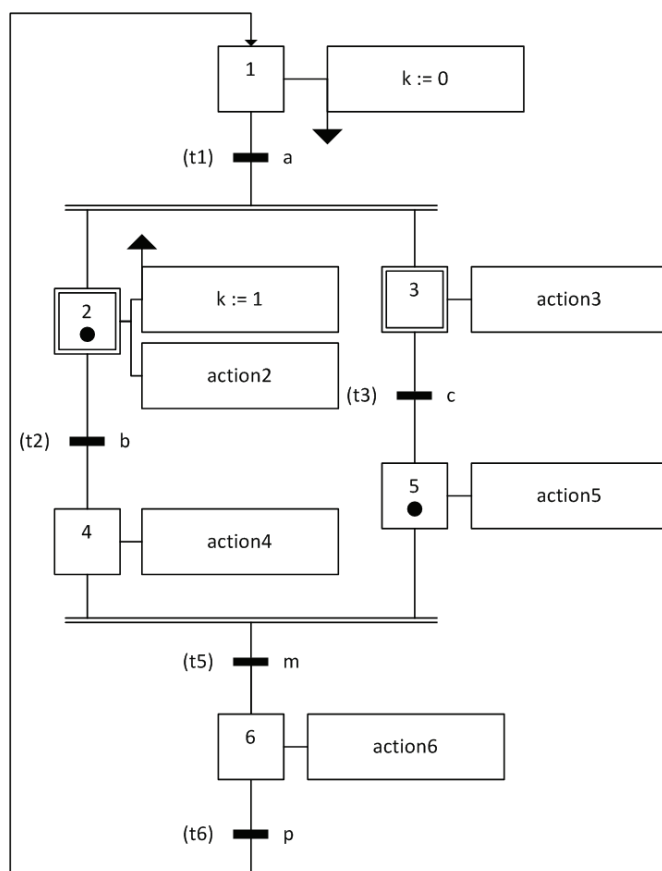


Abbildung 4-3: Beispiel eines typischen Grafcet

wird beispielsweise nur dann ausgeführt ($\text{action3} = \text{true}$), wenn Schritt 3 aktiv ist, für die zugehörige Schrittvariable also $X_3 = \text{true}$ gilt. Sobald Schritt 3 deaktiviert wird, wird auch die Aktion action3 nicht mehr ausgeführt ($\text{action3} = \text{false}$). Die *gespeichert wirkende Aktion* $k := 1$ hingegen wird nur dann ausgeführt, wenn Schritt 2 aktiviert wird, also $\uparrow X_2 = \text{true}$ gilt. Da die IEC 60848 für GRAFCET definiert, dass Zustandsübergänge grundsätzlich keine Zeit in Anspruch nehmen, so auch im Falle der betrachteten Schrittvariablen X_2 , wird die *gespeichert wirkende Aktion* nur einmalig ausgeführt. Die Zuweisung des Wertes 1 zur Variablen k bleibt anschließend so lange bestehen, bis durch eine nachfolgende Aktion k erneut angesprochen wird.

Die unterschiedlichen Aktionsarten bilden demzufolge die Operatoren für die Ausgaben eines Graf cet. Zu den gültigen Operanden in *kontinuierlich wirkenden Aktionen* zählen diejenigen Prozessausgaben und Bedienerückmeldungen des Graf cet, deren Zustand durch die Werte *true* und *false* vollständig beschrieben ist. Auf Seiten der *gespeichert wirkenden Aktionen* sind gültige Operanden alle logischen und numerischen Prozessausgaben, Bedienerückmeldungen und Anweisungen an die Verarbeitungseinheit. Boolesche Ausgaben eines Graf cet können demzufolge entweder über *kontinuierlich wirkende* oder über *gespeichert wirkende Aktionen* angesprochen werden. Um Konflikte bezüglich der Ausgangswertigkeit einer Booleschen Ausgabe zu vermeiden, sollte stets eindeutig festgelegt sein, welche Aktionsart für die jeweilige Ausgabe verwendet wird [DIN EN 60848][#]. Existieren in einer Situation mehrere Schritte, deren *gespeichert wirkende Aktionen* auf dieselbe Ausgangsvariable zugreifen, so besteht ein nicht auflösender Konflikt über den Wert der Ausgangsvariablen. Aus diesem Grund dürfen entsprechende Konflikte in einem Graf cet nicht auftreten. Im Fall von *kontinuierlich wirkenden Aktionen* können solche Konflikte hingegen nicht auftreten. Existieren in einer Situation mehrere Schritte, deren *kontinuierlich wirkende Aktionen* auf dieselbe Boolesche Ausgangsvariable zugreifen, so beträgt der Wert der Ausgangsvariablen *true*.

Durch Transitionsbedingungen und Aktionen ist es möglich, Eingaben und Ausgaben unter Berücksichtigung des internen Zustands der Steuerung in einem Graf cet gezielt miteinander zu verknüpfen. Ein typisches, einfaches Beispiel eines solchen Graf cet zeigt Abbildung 4-3. Für eine Übersicht über alle in GRAFCET möglichen Ausprägungen von Transitionsbedingungen und Aktionen sei an dieser Stelle auf Anhang C dieser Arbeit verwiesen. Weitere Ausführungen finden sich beispielsweise in [DIN EN 60848][#] und Appendix E in [DAVID & ALLA 2010].

4.2.3 Dynamisches Verhalten

Die dynamischen Zusammenhänge des durch Struktur und Wirkungsteil spezifizierten Verhaltens des Ablaufteils einer Steuerung werden durch insgesamt fünf *Ablaufregeln* definiert. Sie geben an, unter welchen Umständen ein Graf cet den Übergang von einer Situation *ante* in eine Situation *post* vollzieht und welche Auswirkungen sich dadurch für die Ausgaben ergeben. Nicht nur bei den grafischen Elementen von GRAFCET wird die enge Verwandtschaft zu Petrinetzen deutlich, sondern auch in den nachfolgend beschriebenen Ablaufregeln.

Die erste Situation, die ein Graf cet einnehmen kann, ist die zum Initialisierungszeitpunkt aktive Situation, welche durch die Menge der Initialschritte definiert ist. Alle weiteren Schritte sind bei Initialisierung in inaktivem Zustand. Hieraus ergibt sich bereits die erste der fünf Ablaufregeln wie folgt:

Regel 1:

Die Initialsituation, gekennzeichnet durch die Menge der Initialschritte, ist die zum Startzeitpunkt aktive Situation eines Graf cet.

Der Übergang von der Initialsituation in die nächste erreichbare Situation erfolgt genau dann, wenn mindestens eine der Transitionen im Nachbereich der aktiven Schritte auslösen kann und zeitgleich alle Schritte in ihrem Vorbereich deaktiviert und alle Schritte in ihrem

Nachbereich aktiviert werden. Bezogen auf eine Transition ergibt sich folgende Regel für ihr Auslösen:

Regel 2:

Eine Transition wird als *freigegeben* bezeichnet, wenn alle mit ihr verknüpften Schritte im Vorbereich aktiviert sind.

Eine Transition *löst aus*, wenn sie freigegeben ist und wenn die mit ihr verknüpfte Transitionsbedingung, in Folge eines Eingangsereignisses, erfüllt ist.

Die nächste erreichbare Situation des Grafcet ist die nächstmögliche *stabile* Situation [DAVID & ALLA 1992]. Eine stabile Situation zeichnet sich dadurch aus, dass die Menge der auslösbaren Transitionen nach dem Übergang in eine Nachfolgesituation und in Folge eines Eingangsereignisses gleich der leeren Menge ist. Die Auswirkungen des Auslösens einer Transition werden in der nachfolgenden Regel 3 zusammengefasst:

Regel 3:

Wenn eine Transition auslöst, so werden alle Schritte im Vorbereich deaktiviert und alle Schritte im Nachbereich aktiviert.

Wie in dem vorangegangenen Unterabschnitt bereits angeführt wurde, benötigt ein Übergang von der Situation *ante* in die Situation *post* aufgrund der algebraischen Definitionen keine Zeit. Dies gilt einerseits für die in Regel 3 definierte Deaktivierung und Aktivierung der Schritte sowie andererseits hinsichtlich der Transitionen, die in Folge desselben Eingangsereignisses gleichzeitig auslösen können:

Regel 4:

Gleichzeitig auslösbare Transitionen werden zeitgleich (simultan) ausgelöst.

Eine Konfliktsituation, wie sie beispielsweise in einem Bedingungs-/Ereignis-Netz [LITZ 2005, S.236] auftreten kann, ist für GRAFCET somit nicht möglich. Es schalten in GRAFCET immer alle auslösbaren Transitionen gleichzeitig und gleichberechtigt, wodurch im Umkehrschluss wiederum ungewünschtes Verhalten auftreten kann, wie beispielsweise bei einer alternativen Verzweigung. Eine Besonderheit ergibt sich, wenn in einer nicht verzweigten Schrittkette Transitionsbedingungen mehrerer aufeinanderfolgender Transitionen aufgrund eines Eingangsereignisses erfüllt sind. Unter Berücksichtigung von Regel 3 kann der Fall eintreten, dass bei einem Übergang von einer stabilen Situation in die nächste stabile Situation einzelne Schritte der Schrittkette übersprungen werden. Im Falle eines solchen *transienten Ablaufs* [DIN EN 60848][#] werden *kontinuierlich wirkende Aktionen*, deren assoziierte Schritte nicht zu der stabilen Situation gehören, nicht ausgeführt. *Gespeichert wirkende Aktionen* werden hingegen ausgeführt.

Die fünfte Ablaufregel definiert das Verhalten eines Grafcet, wenn ein Schritt gleichzeitig deaktiviert und aktiviert wird und berücksichtigt somit das Verhalten in einer Kontaktsituation, wie sie aus dem Bereich der Petrinetze bekannt ist:

Regel 5:

Ein Schritt bleibt aktiv, wenn er durch die Anwendung der Ablaufregeln zeitgleich aktiviert und deaktiviert wird.

Die GRAFCET-Ablaufregeln garantieren deterministisches Verhalten [DAVID 1995] und erlauben es, in Anlehnung an Petrinetze, das Verhalten der Steuerung als eine Art Markenspiel nachzuvollziehen und alle erreichbaren Situationen in einem Zustandsgraphen zusammenzufassen und zu analysieren [ROUSSEL 1994]. Bei einer rechnergestützten Erreichbarkeitsanalyse müssen für die Regeln 3, 4 und 5 allerdings Reihenfolgeregelungen definiert werden, die sich dann in unterschiedlichen Interpretationsalgorithmen niederschlagen, wie beispielsweise in [L'HER ET AL. 1995] oder [EL RHALIBI ET AL. 1994] aufgezeigt.

Somit sind nun die grundlegenden Elemente von GRAFCET sowie deren Bedeutung erläutert worden. Gemäß [DAVID & ALLA 2010] wird dieser Sprachumfang auch als Basic-Grafcet bezeichnet. Alle darüber hinaus in der IEC 60848 definierten Sprachelemente stellen eine Option für eine kompaktere Darstellung spezifischer Sachverhalte dar. Die Anwendung der im nachfolgenden Abschnitt beschriebenen Sprachelemente setzt voraus, dass der Anwender mit den grundlegenden Elementen und Ablaufregeln sowie deren Anwendung vertraut ist. Ein unbedachter Einsatz oder eine übermäßige Verwendung dieser Konstrukte kann schnell zu missverständlichen Spezifikationen führen oder ungewünschtes dynamisches Verhalten eines Grafcet hervorrufen.

4.3 Möglichkeiten zur hierarchischen Strukturierung

4.3.1 Makroschritte

Durch Makroschritte steht ein Sprachelement in GRAFCET zur Verfügung, welches die Komposition bzw. die Dekomposition einzelner Ausschnitte eines Grafcet ermöglicht. Die grafische Repräsentation der Makroschritte entspricht im Grundsatz der eines Schrittes, ergänzt durch zwei horizontal verlaufende Striche innerhalb des Schrittsymbols. Per Definition beginnt der Name eines Makroschrittes mit dem Präfix M. Aktionen sollten mit einem Makroschritt nicht assoziiert werden, da er in der Regel eine Platzhalterfunktion innerhalb des Grafcet einnimmt. Ein Makroschritt ist Platzhalter für einen bestimmten Ausschnitt eines Grafcet, der nachfolgend als Feinstruktur des Makroschrittes bezeichnet wird. Für die Feinstruktur schreibt die IEC 60848 vor, dass es darin jeweils exakt einen Anfangsschritt, gekennzeichnet durch das Präfix E (franz.: Entrée) im Schrittnamen und einen Ausgangsschritt, gekennzeichnet durch das Präfix S (franz.: Sortie) im Schrittnamen, geben muss. Weiterhin muss die Feinstruktur einen eindeutigen Verweis auf den zugehörigen Makroschritt enthalten. Somit eröffnet sich die Möglichkeit, je nach Bedarf des Anwenders, die Feinstruktur an Stelle des Makroschrittes in den Grafcet zu integrieren oder als gesonderten Teil-Grafcet auszulagern und somit in eine grundsätzliche und eine detaillierte Sichtweise zu unterscheiden.

Abbildung 4-4 zeigt einen typischen Grafcet unter der Verwendung eines Makroschrittes M10. Die Feinstruktur von M10 ist im rechten Teil der Abbildung aufgeführt und als Teil-Grafcet ausgelagert. In der Umrahmung der Feinstruktur ist ein eindeutiger Verweis auf den zugehörigen Makroschritt durch Angabe des Makroschrittnamens und in Übereinstimmung mit den Vorgaben des Standards enthalten. Es können mehrere Duplikate eines Makroschrittes innerhalb eines Grafcet verwendet werden und auf dieselbe Feinstruktur verweisen. Ersetzt man die Duplikate des Makroschrittes durch die Feinstruktur, so wird für jeden Makroschritt eine Instanz der Feinstruktur integriert. Hinsichtlich der Auswirkungen der Verwendung von Makroschritten auf das dynamische Verhalten müssen zwei Fälle unterschieden

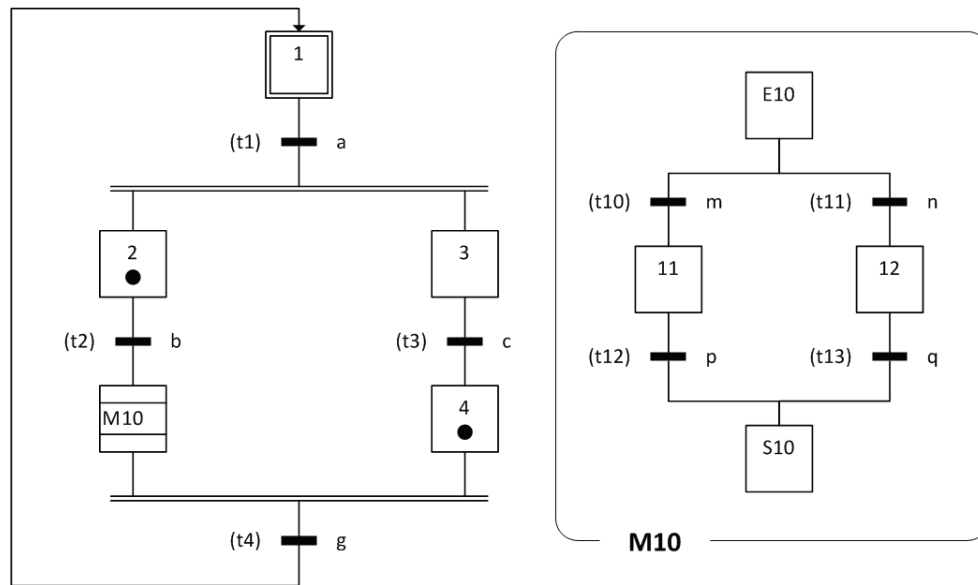


Abbildung 4-4: Struktur eines Grafcet mit Makroschritt

werden. Im ersten Fall wird ein Makroschritt in einem Grafcet verwendet, es existiert allerdings noch keine Feinstruktur. Dies ist beispielsweise in einer frühen Phase des Steuerungsentwurfs denkbar, wenn zunächst das grobe Verhalten der Steuerung modelliert werden soll. Hier kann der Makroschritt als Schritt betrachtet werden, mitsamt allen zugehörigen Eigenschaften.

Im zweiten Fall existiert eine Feinstruktur, wie im Beispiel aus Abbildung 4-4. Hier verhält sich der Makroschritt nicht wie ein Schritt. Wird in Folge eines Ereignisses der Makroschritt aktiviert, beispielsweise durch Auslösen einer Transition im Vorbereich, so wird gleichzeitig der Eingangsschritt der Feinstruktur aktiviert. In den nachfolgenden Situationen muss zunächst die Feinstruktur durchlaufen werden, bevor der Makroschritt wieder deaktiviert wird, denn die Transitionen im Nachbereich des Makroschritts sind erst dann freigegeben, wenn der Ausgangsschritt der Feinstruktur aktiv ist. Somit ergeben sich implizite Abhängigkeiten zwischen Makroschritt und Feinstruktur, die in der Anwendung der Ablaufregeln berücksichtigt werden müssen. Die Ablaufregeln selbst müssen im Falle von Makroschritten allerdings nicht modifiziert werden.

In der gegebenen Situation des Grafcet in Abbildung 4-4 erfolgt der Übergang in die nächstmögliche Situation genau dann, wenn in Folge eines Ereignisses Transition t_2 auslöst. Da Transition t_2 bereits freigegeben ist (Schritt 2 aktiv), löst sie genau dann aus, wenn die Eingangsvariable b ihren Wert von *false* in *true* ändert. Zeitgleich mit dem Auslösen von t_2 wird Schritt 2 deaktiviert und Makroschritt M10 aktiviert, wodurch der Eingangsschritt der Feinstruktur E10 ebenfalls aktiviert wird. In dieser Situation ist Transition t_4 nicht freigegeben. Transition t_4 ist erst dann freigegeben, wenn der Ausgangsschritt der Feinstruktur S10 aktiv ist.

4.3.2 Einschließende Schritte

Das Prinzip der Einschließung [GUÉGUEN ET AL. 1999, GUÉGUEN & BOUTEILLE 2001] bietet die Möglichkeit, unidirektionale Abhängigkeiten zwischen zunächst unabhängig voneinander spezifizierten Graf cets, sogenannten Teil-Graf cets, in kompakter Form festzulegen, und ist seit 2002 fester Bestandteil der IEC 60848. Es ist somit möglich, eine hierarchische, nicht-rekursive Struktur in einem Graf cet zu definieren, wobei sowohl eine top-down als auch eine bottom-up Vorgehensweise unterstützt wird. Auf der obersten Hierarchieebene existiert ein einziger, übergeordneter Teil-Graf cet, der als globaler Graf cet bezeichnet wird. Eine Einschließung besteht aus einem *einschließenden Schritt*, der Teil des übergeordneten Teil-Graf cets ist, und dem sogenannten Teil-Graf cet der eingeschlossenen Schritte, welcher die untergeordnete Ebene repräsentiert. Der Teil-Graf cet der eingeschlossenen Schritte kann wiederum *einschließende Schritte* enthalten, die eine Abhängigkeit zu untergeordneten Teil-Graf cets festlegen. Für den untergeordneten Teil-Graf cet der eingeschlossenen Schritte können Schritte explizit bestimmt werden, die zeitgleich mit Aktivierung des *einschließenden Schrittes* aktiviert werden, wie im Folgenden erläutert wird. Ein *einschließender Schritt* besitzt grundsätzlich alle Eigenschaften eines Schritts und wird durch ein spezielles Schrittsymbol dargestellt, zum Beispiel Schritt 2 in Abbildung 4-5. Ein eingeschlossener Schritt, also ein von einer Einschließung betroffener Schritt, ist ein Schritt-Element des Teil-Graf cets der eingeschlossenen Schritte und wird grafisch durch das Asteriskus-Symbol * links neben dem Schritt gekennzeichnet. Der Teil-Graf cet der eingeschlossenen Schritte enthält einen eindeutigen Verweis auf den zugehörigen *einschließenden Schritt*. Im Fall des Teil-Graf cets G10 in Abbildung 4-5 wird am linken oberen Rand auf den einschließenden Schritt 2 verwiesen

In Bezug auf die Auswirkungen auf das dynamische Verhalten eines Graf cet ist festzuhalten, dass die fünf Ablaufregeln weiterhin uneingeschränkt gültig sind und nicht modifiziert werden müssen. Wird ein *einschließender Schritt* aktiviert, so werden gleichzeitig die mit ihm verknüpften eingeschlossenen Schritte aktiviert, zum Beispiel Schritt 11 in Abbildung 4-5. Solange der *einschließende Schritt* aktiv ist, unterliegt der von der Einschließung betroffene Teil-Graf cet den fünf Ablaufregeln. Wird der *einschließende Schritt* deaktiviert, beispielsweise durch das Auslösen einer Transition in dessen Nachbereich, so werden gleichzeitig alle Schritte des Teil-Graf cets der eingeschlossenen Schritte deaktiviert und erst dann wieder im dynamischen Ablauf berücksichtigt, wenn die Einschließung erneut aktiv ist. Enthält der Teil-Graf cet der eingeschlossenen Schritte keinen Initialschritt, so sind bei Initialisierung des Graf cet alle Schritte inaktiv. Enthält der Teil-Graf cet der eingeschlossenen Schritte mindestens einen Initialschritt, so muss der zugehörige *einschließende Schritt* zwangsläufig ein Initialschritt sein (Symbol für *einschließenden Schritt* mit doppelter Umrahmung). Ein Teil-Graf cet der eingeschlossenen Schritte kann aus mehreren einzelnen Teil-Graf cets bestehen. Umgekehrt kann allerdings ein Teil-Graf cet der eingeschlossenen Schritte nur einem *einschließenden Schritt* zugeordnet sein.

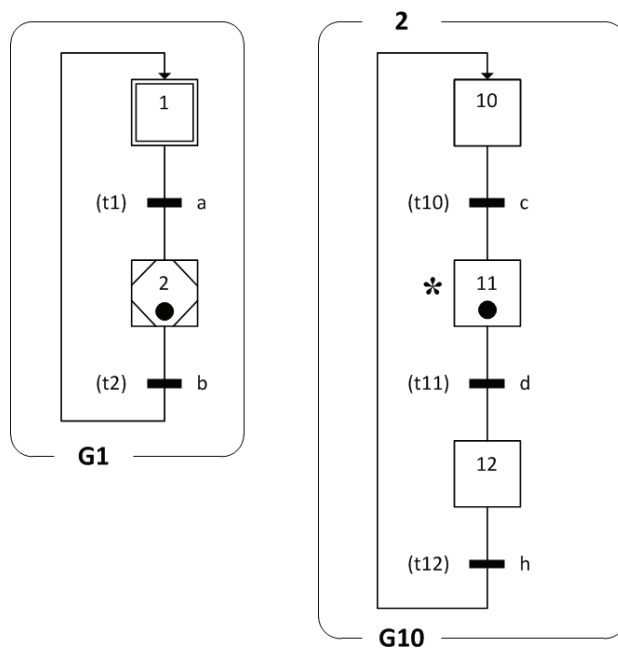


Abbildung 4-5: Struktur eines Grafcet mit einschließendem Schritt

Abbildung 4-5 zeigt die Struktur eines Grafcet, bestehend aus zwei Teil-Grafcets (G1 und G10), die durch eine Einschließung verknüpft sind. Die hierarchische Struktur ergibt sich aus den eindeutigen Kennzeichnungen in den Umrahmungen der Teil-Grafcets. So verweist G10 auf den *einschließenden Schritt 2* des übergeordneten Teil-Grafcet G1. In der Initialsituation waren alle Schritte mit Ausnahme von Schritt 1 inaktiv. Durch Auslösen von Transition t_1 resultiert die momentan aktive Situation, in der der *einschließende Schritt 2* und zeitgleich der eingeschlossene Schritt 11 aktiv sind. G10 unterliegt von nun an uneingeschränkt den fünf Ablaufregeln, so dass das Auslösen von Transition t_{11}

zur Deaktivierung von Schritt 11 und zur Aktivierung von Schritt 12 führt. Löst allerdings Transition t_2 aus, so werden alle Schritte von G10 unmittelbar deaktiviert.

4.3.3 Zwangssteuernde Befehle

Eine weitere Möglichkeit, unidirektionale Abhängigkeiten zwischen Teil-Grafcets festzulegen, ist durch die Verwendung *zwangssteuernder Befehle* gegeben. Als Ergänzung zu den bereits diskutierten Makroschritten (\rightarrow 4.3.1) wurden diese zunächst als Makroaktionen in [DENEUX 1983] definiert, in [DAVID & ALLA 1992] einer breiteren Öffentlichkeit zugänglich gemacht und unter Forschern ausgiebig diskutiert [LESAGE & ROUSSEL 1993, AZEVEDO & ESTIMA DE OLIVEIRA 1999]. Der Großteil der in [DENEUX 1983] definierten Makroaktionen in GRAFCET wurde schließlich unter dem Begriff der *zwangssteuernden Befehle* in die überarbeitete Version der Norm aus dem Jahre 2002 aufgenommen. Im Rahmen dieser Arbeit werden nachfolgend nur die in IEC 60848 definierten *zwangssteuernden Befehle* erläutert. Für eine weiterführende Diskussion des Konzepts der Makroaktionen, insbesondere des Prinzips *force*, sei an dieser Stelle auf [DAVID & ALLA 2010] verwiesen.

Ein *zwangssteuernder Befehl* in GRAFCET wird in Anlehnung an das Symbol für *kontinuierlich wirkende Aktionen* mit einer doppelten Umrahmung grafisch dargestellt. Ein Beispiel ist in Abbildung 4-6 ersichtlich. Eine horizontale Verbindung assoziiert den *zwangssteuernden Befehl* mit einem Schritt. Wie bei *kontinuierlich wirkenden Aktionen* wird die durch den *zwangssteuernden Befehl* spezifizierte Zwangssteuerung so lange ausgeführt, wie der assoziierte Schritt im aktiven Zustand ist. Im Gegensatz zu *kontinuierlich wirkenden Aktionen*, in denen Ausgaben die Operanden sind, handelt es sich bei den Operanden einer Zwangssteuerung um Teil-Grafcets. Diese werden in ihrem

dynamischen Verhalten so eingeschränkt, dass sie in einer definierten Situation verharren müssen und im Sinne der Ablaufregeln als eingefroren gelten.

Die Definitionen der IEC 60848 umfassen die folgenden definierten Situationen:

- $G\#\{\text{INIT}\}$ \Leftrightarrow Zwangssteuerung von Teil-Grafcet # in die Initialsituation,
- $G\#\{*\}$ \Leftrightarrow Zwangssteuerung von Teil-Grafcet # in die momentane Situation,
- $G\#\{\}$ \Leftrightarrow Zwangssteuerung von Teil-Grafcet # in die leere Situation,
- $G\#\{13, 22\}$ \Leftrightarrow Zwangssteuerung von Teil-Grafcet # in die bestimmte Situation.

Wie auch bei dem Prinzip der Einschließung, besteht durch Zwangssteuerungen die Möglichkeit, nicht-rekursive hierarchische Strukturen in GRAFCET festzulegen. Die durch einen *zwangssteuernden Befehl* verkoppelten Teil-Grafcets sind in Bezug auf ihr dynamisches Verhalten aber grundsätzlich voneinander unabhängig. Abbildung 4-6 zeigt hierzu ein einfaches Beispiel eines *zwangssteuernden Befehls* in Teil-Grafcet G1, welcher den untergeordneten Teil-Grafcet G10

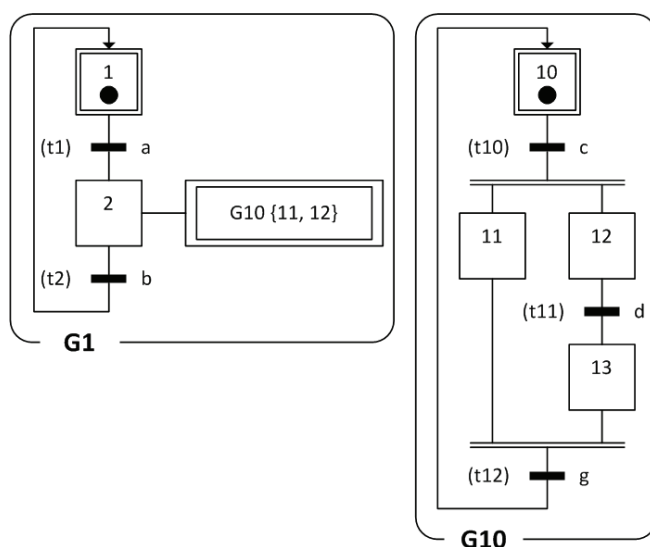


Abbildung 4-6: Struktur eines Grafcet mit zwangssteuerndem Befehl

in die bestimmte Situation $\{11, 12\}$ zwingt. Die momentane Situation ist die Initialsituation des Grafcet, in der nur die Schritte 1 und 10 aktiv sind. Führt nun ein Ereignis zum Auslösen von Transition t_1 , so wird Schritt 1 deaktiviert und Schritt 2 aktiviert. Mit Aktivierung des Schritts 2 wird auch der assoziierte *zwangssteuernde Befehl* $G10\{11, 12\}$ ausgeführt, so lange Schritt 2 aktiv ist. Durch die Ausführung der Zwangssteuerung werden alle Schritte von G10 deaktiviert und nur die zur definierten Situation zugehörigen Schritte aktiviert, wie beispielsweise Schritt 11 und 12 in Abbildung 4-6. In dieser definierten Situation verharrt der zwangsgesteuerte Teil-Grafcet G10, bis ein Ereignis eintritt, in dessen Folge Schritt 2 deaktiviert wird. Ausgehend von der definierten Situation der Zwangssteuerung gelten von nun an wieder die fünf Ablaufregeln.

Während der Ausführung der Zwangssteuerung besitzt der *zwangssteuernde Befehl* Priorität gegenüber den fünf Ablaufregeln, so dass die Ablaufregeln für den zwangsgesteuerten Teil-Grafcet nicht angewendet werden können. Im Falle von *zwangssteuernden Befehlen* ergibt sich somit eine zusätzliche Ablaufregel:

Regel 6:

Ein aktiver *zwangssteuernder Befehl* besitzt stets Vorrang vor den Ablaufregeln 1-5, in der Weise, dass der zwangsgesteuerte Teil-Grafcet als eingefroren gilt, so lange die Zwangssteuerung aktiv ist.

4.4 Zeitabhängige Bedingungen

Die in den vorangehenden Kapiteln diskutierten GRAFCET-Elemente ermöglichen eine übersichtliche grafische Modellierung des Steuerungsablaufs, also eine Spezifikation dessen, was die Steuerung in welcher Folge tun soll. Der Zeitpunkt für einen Übergang eines Grafcet in eine neue Situation und die Zeitdauer für die Ausführung von Aktionen, in Abhängigkeit von den jeweils aktivierten Schritten, kann mit den bisher bekannten GRAFCET-Elementen aber nur implizit spezifiziert werden. So können Ereignisse und Bedingungen in Transitions- und Zuweisungsbedingungen definiert werden, die Bedienerangaben, Prozessrückmeldungen und Rückgaben der Verarbeitungseinheit logisch miteinander verknüpfen. Zeitpunkt oder Zeitdauer ergeben sich implizit aus der Struktur des Grafcet in Kombination mit der Reihenfolge der Eingaben. Um zeitliche Aspekte auch explizit in einem Grafcet angeben zu können, definiert die aktuell gültige Norm IEC 60848 *zeitabhängige Bedingungen*, die in Transitions- und Zuweisungsbedingungen Verwendung finden können. Die Berücksichtigung von Zeit als explizite, diskrete Größe im Sinne ereignisdiskreter Systeme wird so in GRAFCET möglich. Dies bedeutet, dass keine globale Uhrzeit existiert, die sich als kontinuierliche Größe auf das dynamische Verhalten eines Grafcet auswirkt. Vielmehr handelt es sich bei der Zeit in Bezug auf GRAFCET um eine verteilte lokale Größe, die jeweils in direkter Abhängigkeit zu diskreten Ereignissen steht. Im Folgenden sollen die wesentlichen Merkmale *zeitabhängiger Bedingungen* gemäß [DIN EN 60848][#] erläutert werden.

Bei einer *zeitabhängigen Bedingung* handelt es sich um einen logischen Ausdruck, der aus einer zeitabhängigen Variablen und zwei Zeitangaben besteht. Die grundsätzliche Notation lautet

$$t_1/x/t_2, \quad (4-1)$$

wobei die Variablen t_1 und t_2 die Zeitangaben repräsentieren und x als Platzhalter für die zeitabhängige Variable dient. Eine Zeitangabe besteht stets aus einem numerischen Wert und der zugehörigen Zeiteinheit, wie beispielsweise $t_1=3 \text{ min}$. Als zeitabhängige Variable sind laut IEC 60848 sämtliche logischen Ausdrücke (\rightarrow 4.2.2), ausschließlich der Ausdrücke *true* und *false*, zulässig. Eine gültige *zeitabhängige Bedingung* wäre demnach:

$$3 \text{ s} / a / 5 \text{ s} . \quad (4-2)$$

In Bezug auf das dynamische Verhalten von GRAFCET wird die *zeitabhängige Bedingung* wie eine Bedingung ausgewertet. Sie ist erfüllt ($[t_1/x/t_2] = \text{true}$), wenn die zeitabhängige Variable x ihren Zustand von *false* in *true* ändert ($\uparrow x$) und die Zeit t_1 abgelaufen ist. Sie bleibt erfüllt bis zu dem Zeitpunkt, zu dem die zeitabhängige Variable ihren Zustand von *true* in *false* ändert ($\downarrow x$) und die Zeit t_2 abgelaufen ist. In allen anderen Fällen ist die Bedingung nicht erfüllt ($[t_1/x/t_2] = \text{false}$). Am Beispiel aus Formel (4-2) bedeutet dies, dass die *zeitabhängige Bedingung* drei Sekunden nach dem Ereignis ($\uparrow a$) den logischen Wert *true* annimmt. Erst fünf Sekunden nach dem nachfolgenden Ereignis ($\downarrow a$) ändert sich der logische Wert der *zeitabhängigen Bedingung* wieder in *false*. Abbildung 4-7 verdeutlicht die zuvor beschriebenen Zusammenhänge noch einmal in Form eines Diagramms für die *zeitabhängige Bedingung* aus Formel (4-2).

Eine verkürzte Art der Notation ergibt sich für den Fall, dass $t_2 = 0$ gilt. In diesem Fall ist auch die folgende Schreibweise zulässig:

$$t_1/x \tag{4-3}$$

Durch die bisher aufgezeigten *zeitabhängigen Bedingungen* können zeitliche Verzögerungen spezifiziert werden. Darüber hinaus sieht der IEC 60848 Standard aber auch die Spezifikation von zeitlichen Limitierungen vor, welche beispielsweise zur Angabe einer zeitbegrenzten Ausführung *kontinuierlich wirkender Aktionen* verwendet werden können. Hierfür wird eine spezielle Notation eingeführt

$$\overline{t_1/x} \tag{4-4}$$

Diese *zeitabhängige Bedingung* ist erfüllt ($[\overline{t_1/x}] = true$), wenn die zeitabhängige Variable x ihren Zustand von *false* in *true* ändert ($\uparrow x$). Sie bleibt solange erfüllt bis die Zeit t_1 abgelaufen ist. In allen anderen Fällen ist sie nicht erfüllt.

Zeitabhängige Bedingungen können als Transitionsbedingungen (Nr. 5-6, → Anhang C) oder

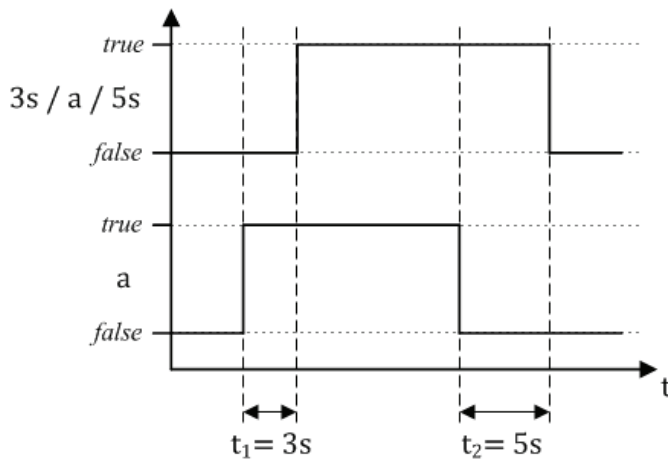


Abbildung 4-7: Semantik der zeitabhängigen Bedingung aus Formel (4-2)

als Zuweisungsbedingungen von *kontinuierlich wirkenden Aktionen* (Nr. 9, → Anhang C) in einem Grafcet verwendet werden. Im Falle von Transitionsbedingungen definiert der IEC 60848 Standard die Notationen aus (4-1) und (4-3) als gültige zeitabhängige Transitionsbedingungen. Für *kontinuierlich wirkende Aktionen* sieht die Norm die Notation aus (4-1) uneingeschränkt, sowie die Notationen aus (4-3) und (4-4) unter ausschließlicher Verwendung der Schrittvariablen des assoziierten

Schrittes als zeitabhängige Bedingungen vor. Weitere Informationen hinsichtlich zeitabhängiger Transitions- und Zuweisungsbedingungen sind einerseits in ausführlicher Art und Weise in [SCHUMACHER & FAY 2013 A]* gegeben, andererseits in Anhang C aufgeführt.

In der überarbeiteten, dritten Fassung des IEC 60848 Standards gibt es eine weitere Möglichkeit, *zeitabhängige Bedingungen* in GRAFCET zu spezifizieren. So sieht die dritte Ausgabe des Standards IEC 60848 [IEC 60848 B]# in Ergänzung zur Schrittvariablen X die Definition einer Stoppuhr-Variablen T für jeden Schritt vor. Sie gibt diejenige Zeit zurück, die seit der Aktivierung des Schrittes vergangen ist, und stellt eine alternative Schreibweise zu (4-3) dar:

$$5s/X_{10} \equiv T_{10} \geq 5s. \tag{4-5}$$

Im Vergleich zu [IEC 60848 A]# wurden, mit Ausnahme der Einführung der Stoppuhr-Variablen T , inhaltlich keine weiteren Änderungen in [IEC 60848 B]# vorgenommen.

4.5 Werkzeugunterstützung

Sowohl im Bereich der Forschung, als auch auf kommerzieller Ebene hat das Beschreibungsmittel GRAFCET Einzug in unterschiedlichste Werkzeuge gefunden, die das Erstellen eines Grafcet unterstützen. Insbesondere im Bereich der Forschung ist eine viel diskutierte Fragestellung, wie die Erstellung eines Grafcet unter Nutzung eines softwarebasierten Werkzeugs methodisch unterstützt werden kann. Die im Zusammenhang mit GRAFCET entwickelte GEMMA-Methodik wird häufig diskutiert [FREY & KLEIN 2004]. Die gemeinsame Betrachtung von GEMMA und GRAFCET geht über den Rahmen dieser Arbeit hinaus, so dass an dieser Stelle beispielsweise auf [PONSA ET AL. 2009, MACHADO & SEABRA 2010] oder [ALVAREZ ET AL. 2012] verwiesen wird. Dieser Abschnitt gibt einen kurzen Überblick über die wichtigsten verfügbaren, softwarebasierten Werkzeuge für Grafcet und diskutiert deren charakteristische Vorzüge und Nachteile.

JGrafchart [JGRAFCHART][@] ist eine Entwicklung des *Department of Automatic Control* der schwedischen *LUND Universität* [LUND][@] und als Software frei verfügbar. Der auf der *Java-Technologie* [JAVA][@] basierende Editor unterstützt die Erstellung von Spezifikationen mit dem Beschreibungsmittel Grafchart [JOHNSON & ÅRZÉN 1999]. Grafchart basiert auf GRAFCET und bietet weiterführende spezifische Sprachelemente. Grundsätzlich unterstützt *JGrafchart* GRAFCET vollumfänglich und bietet darüber hinaus umfangreiche Funktionalitäten, darunter auch eine Im- und Export-Schnittstelle auf der Basis von XML [W3C XML 1.0][#]. Allerdings handelt es sich hierbei um ein proprietäres XML-Schema, durch das lediglich grafische Informationen des *JGrafchart*-Editors ausgetauscht werden können. GRAFCET-spezifische Informationen werden hingegen nicht hinterlegt, so dass eine weitere Verwendung im Zuge formaler Methoden nicht möglich ist. Abbildung 4-8 zeigt die Benutzeroberfläche von *JGrafchart* mit dem Beispiel eines Grafcet.

FluidSim [FLUIDSIM][@], eine Entwicklung des Unternehmens *Art Systems*, dient vorrangig dem Erstellen und Simulieren elektropneumatischer und -hydraulischer Schaltungen und wird

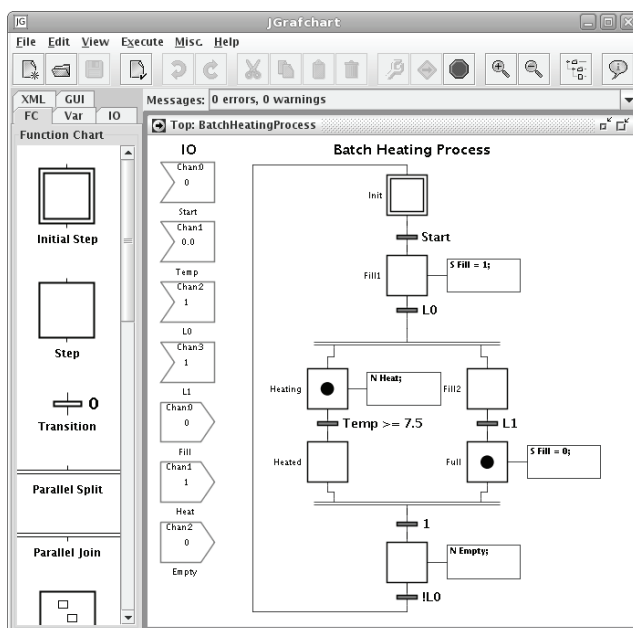


Abbildung 4-8: Benutzeroberfläche von *JGrafchart* (Screenshot), entnommen aus [JGrafchart][@]

in Verbindung mit den Lern- und Ausbildungssystemen der *Festo Didactic GmbH & Co. KG* vertrieben. Die Software ist in einer Testversion frei verfügbar und ermöglicht die Erstellung von Grafcets zur Ablaufbeschreibung der konfigurierten Schaltung. Über eine herstellerspezifische Datenschnittstelle besteht die Möglichkeit, das Verhalten der Schaltung anhand einer Simulation oder der realen Schaltung nachzuvollziehen. Es ist jedoch nicht möglich, einen Grafcet unabhängig von der technischen Realisierung der jeweiligen Schaltung zu erstellen. GRAFCET spielt in diesem Werkzeug eine eher untergeordnete Rolle, obwohl umfangreiche

Lernunterlagen zur Verfügung stehen (siehe z. B. [FESTO 2008]). GRAFCET-spezifische Informationen können zudem nicht in einem offenen Datenaustauschformat abgespeichert werden.

WinErs GRAFCET ist eine Software der *Ingenieurbüro Schoop GmbH* und für praxisorientierte Lernzwecke konzipiert [WINERS][®]. Eine kostenlose Testversion ist verfügbar. Ein Graf cet kann durch automatische Prüfrou tinen auf seine syntaktische Korrektheit hin untersucht und auch in seinem dynamischen Verhalten simuliert werden. In der kostenpflichtigen Version kann zusätzlich die Kopplung des spezifizierten Steuerungsablaufs über eine proprietäre Schnittstelle mit einer Demonstratoranlage erfolgen. Hierbei wird der erstellte Graf cet nicht kompiliert und in den Programmspeicher einer Steuerung übertragen, sondern die an der Anlage zur Verfügung stehenden Ein- und Ausgangsvariablen werden über eine proprietäre Datenschnittstelle ausgetauscht. Wie im Falle von *FluidSim* können auch bei *WinErs GRAFCET* keine GRAFCET-spezifischen Informationen in einem offenen Datenformat abgespeichert werden.

Der Name der Software *SFCedit* [SFCEDIT][®] erscheint zunächst irreführend. Es handelt sich aber auch hierbei um einen GRAFCET-Editor, der alle in IEC 60848 definierten Elemente unterstützt. Der Schwerpunkt dieses Werkzeugs liegt auf der Erstellung und Simulation von Graf cets. Die Kopplung mit realen Hardwarekomponenten einer Anlage ist in diesem Fall nicht vorgesehen. Der Editor stellt alle grundlegenden Funktionen bereit und ermöglicht die Speicherung des Graf cet im XML-Datenformat. Wie in [ALVAREZ ET AL. 2012] aufgezeigt wird, handelt es sich hierbei um ein proprietäres XML-Schema, welches neben grafischen Informationen auch GRAFCET-spezifische Informationen berücksichtigt. Aufgrund der fehlenden Rückverfolgbarkeit des XML-Schemas ist dessen Auswertung allerdings mit Unsicherheiten verbunden und nur mit viel Aufwand vollständig nachvollziehbar.

4.6 Zwischenfazit

Inspiriert durch Petrinetze, ist GRAFCET, nicht zuletzt durch seine Möglichkeiten zur hierarchischen Strukturierung, ein mächtiges grafisches Beschreibungsmittel zur Spezifikation von Steuerungsabläufen. GRAFCET abstrahiert von einer konkreten Realisierung des Ablaufteils der Steuerung. Im Gegensatz dazu stellen die aus GRAFCET hervorgegangenen und grafisch sehr ähnlichen Sequential Function Charts die Struktur eines Programms für speicherprogrammierbare Steuerungen dar und sind selbst eine in IEC 61131-3 definierte grafische Programmiersprache [DIN EN 61131-3][#]. Zwar existieren auch Ansätze in der Forschung, GRAFCET als Programmiersprache zu interpretieren und anzuwenden, wie beispielsweise in den Beiträgen von [WALLÉN 1995], [MARCÉ ET AL. 1996] oder [BAYÓ-PUXAN 2008] beschrieben wird, allerdings wird hierbei lediglich eine Teilmenge des Sprachumfangs der IEC 60848 betrachtet und GRAFCET als Synonym für SFC verwendet. Aus der engen Verwandtschaft von GRAFCET und SFC darf allerdings nicht geschlussfolgert werden, dass beide Begriffe synonym sind [ROUSSEL ET AL. 1999]. Eine detaillierte Analyse über die Unterschiede hinsichtlich Struktur und dynamischem Verhalten findet sich beispielsweise in [SCHUMACHER & FAY 2011]^{*}. Inwieweit es möglich ist, einen Graf cet vollumfänglich als SFC zu implementieren, wird im Rahmen des in dieser Arbeit vorgestellten Transformationskonzeptes in den nachfolgenden Kapiteln noch im Detail diskutiert werden.

Die IEC 61131-3 und somit auch SFCs wurden von Beginn an als internationale Norm konzipiert und definiert, wohingegen GRAFCET zunächst auf nationaler Ebene durch französische Normungsgremien und Fachausschüsse beschrieben wurde. Diesen Aspekt führen auch [BAKER ET AL. 1987] als Vorteil der IEC 61131-3 an, die darauf hinweisen, dass die direkte internationale Normung von SFCs als Teil der IEC 61131-3, zu einer höheren Akzeptanz auf Seiten der Anwender führen wird. Lediglich in Frankreich wird GRAFCET eine weitreichende Akzeptanz in Forschung und Anwendung bescheinigt. Dieses Bild zeigt sich auch heute noch in Bezug auf die Anwendung von GRAFCET und SFC im Steuerungsentwurf.

Vor allem in Frankreich ist GRAFCET bis heute ein weit verbreitetes Beschreibungsmittel in Forschung und Anwendung, wie beispielsweise durch [PROVOST ET AL 2010 A] bestätigt wird. Im internationalen Umfeld hingegen dominiert insbesondere die IEC 61131-3 und findet große Akzeptanz bei der Programmierung von SPSen. SFCs sind zwar weitreichender Gegenstand der Forschung [WIGHTKIN ET AL. 2011], in der praktischen Anwendung zur Programmierung von SPSen sind aber auch sie noch nicht die bevorzugte Programmiersprache [HAJARNAVIS & YOUNG 2008]. Ein Blick auf die deutsche Automatisierungslandschaft zeigt, dass auch hier die IEC 61131-3 in Industrie und Forschung weit verbreitet ist, nicht zuletzt aufgrund der weitreichenden Werkzeugunterstützung. GRAFCET findet erst in jüngster Vergangenheit einen Platz als verbindlicher Ausbildungsinhalt in den Lehrplänen berufsbildender Schulen [TS-HANNOVER 2010, LP-RHEINLAND-PFALZ 2011][@], hat in der industriellen Anwendung momentan allerdings keine Relevanz. Ein Grund hierfür ist beispielsweise die unzureichende Unterstützung durch Werkzeuge, die heutzutage vorrangig didaktischen Zwecken dienen (→ 4.5).

In Bezug auf das Beschreibungsmittel GRAFCET existieren einige bedeutende Forschungsansätze, um formale Methoden im Steuerungsentwurf (→ 2) mit diesem Beschreibungsmittel zu etablieren. Da die Wurzeln von GRAFCET in Frankreich liegen und eine entsprechende Steuerungsspezifikation sowohl in der industriellen Anwendung als auch im Bereich der Forschung weit verbreitet und akzeptiert ist, sind entsprechende wissenschaftliche Impulse weitestgehend durch französische Wissenschaftlerinnen und Wissenschaftler erfolgt und veröffentlicht worden. Im nachfolgenden Kapitel 5 soll ein Überblick über den Stand der Forschung hinsichtlich GRAFCET gegeben werden.

5 Stand der Forschung und Handlungsbedarf bezüglich GRAFCET

5.1 Stand der Forschung

5.1.1 Ansätze zur Verifikation und Validierung

Eine für die formale Verifikation und Validierung bedeutende Arbeit stellt [ROUSSEL 1994] dar, in der das dynamische Verhalten von GRAFCET formal als Zustandsautomat beschrieben wird. Hierzu wird eine spezifische, erweiterte Boolesche Algebra definiert, auf deren Basis ein Grafcet algorithmengestützt direkt in einen verhaltensäquivalenten Zustandsautomaten überführt und analysiert werden kann. Der als Erreichbarkeitsgraph bezeichnete endliche Zustandsautomat umfasst alle stabilen Situationen eines Grafcet in Form von Zuständen und zugehörige Zustandsübergänge, die in direkter Beziehung zu den Transitionsbedingungen des Grafcet stehen. Ähnliche Untersuchungen wurden zuvor für *autonome Grafquets*, also Grafquets ohne Ein- und Ausgaben, in [BLANCHARD 1979] veröffentlicht. Der Erreichbarkeitsgraph bildet eine Art Bindeglied zwischen der Petrinetz-basierten und der Automaten-basierten Sichtweise auf das GRAFCET-Verhalten [PROVOST ET AL. 2010 A] und ist die Grundlage für zahlreiche Forschungsansätze zur formalen Verifikation und Validierung.

Aufgrund der historisch begründeten Abstammung von Petrinetzen (\rightarrow 4.1) ist es zunächst naheliegend, GRAFCET formal als eine spezifische Klasse von Petrinetzen zu beschreiben [DAVID & ALLA 2010], welche eine Folge von Booleschen Eingaben zustandsabhängig in eine Folge von Booleschen Ausgaben umwandelt. SIPN, wie beispielsweise in [KÖNIG & QUÄCK 1988] oder [FREY 2002] beschrieben, bieten hierfür eine geeignete mathematische Basis und eröffnen die Möglichkeit weiterführender methodischer Ansätze zur formalen Analyse. Eine systematische Vorgehensweise zur Überführung eines Grafcet in ein zunächst strukturgleiches und anschließend verhaltensgleiches Petrinetz wird beispielsweise durch [EL RHALIBI ET AL. 1995] vorgeschlagen. Auf der Grundlage der erreichbaren Markierungen des verhaltensgleichen Petrinetzes (Erreichbarkeitsgraph des Petrinetzes) können anschließend Analysen bezüglich der Erreichbarkeit einzelner Markierungen oder Verklemmungen (Deadlocks) algorithmengestützt erfolgen. Somit sind auch entsprechende qualitative Aussagen für die Eigenschaften eines Grafcet möglich. Allerdings berücksichtigen die Autoren lediglich eine Untermenge der Elemente, welche die IEC 60848 für GRAFCET vorsieht. So werden Aktionen, *zeitabhängige Bedingungen* und Konstrukte zur hierarchischen Strukturierung nicht berücksichtigt, wodurch lediglich die Transformation einfacher Schrittketten möglich ist.

Weiterführende Forschungsansätze, wie beispielsweise [PROVOST ET AL. 2010 A], fokussieren darauf, nicht nur strukturelle Eigenschaften eines Grafcet formal zu analysieren, sondern darüber hinaus das dynamische Verhalten gegen Anforderungen an die Steuerung oder das Verhalten der Strecke zu testen. Die Tests können grundsätzlich in Verbindung mit einem formalen Modell der ungesteuerten Strecke oder ohne ein entsprechendes Modell erfolgen. Im letztgenannten Fall resultieren die möglichen Eingaben in das Steuerungsmodell nicht aus dem zu steuernden System (Strecke), sondern werden in Form von manuell oder automatisch generierten Testfällen geprüft. Jeder Testfall enthält eine Kombination aus Eingaben, die

mögliche Zustände der Strecke repräsentieren. Die Reihung von Testfällen ergibt Testsequenzen, die insgesamt alle kombinatorischen Möglichkeiten für Eingaben berücksichtigen sollten, also idealerweise vollständig sind [CHÉRIAUX ET AL. 2010]. Das Eingabe-/ Ausgabeverhalten des Modells der Steuerung wird somit überprüft und die durch den Grafcet generierte Ausgabefolge mit der erwarteten verglichen (→ Abbildung 5-1). Stimmen die durch das Modell der Steuerung generierte und die erwartete Ausgabefolge überein, so erfüllt das Modell der Steuerung die Steuerungsaufgabe. Sollten sich Abweichungen ergeben, so weist das Steuerungsmodell Fehler auf, die erkannt und beseitigt werden können, bevor die Implementierung erfolgt.

Wie in [PROVOST ET AL. 2009, 2011 B] erläutert, ist die formale Definition von GRAFCET als SIPN sowie die automatische Generierung des Erreichbarkeitsgraphen der erste Schritt in der systematischen Vorgehensweise. Die in [PROVOST ET AL. 2010 A] aufgezeigte Methode, den Erreichbarkeitsgraphen eines Grafcet in einen Mealy-Zustandsautomaten zu transformieren, eröffnet in der Folge, bereits bewährte Methoden zur Generierung von Testsequenzen für Mealy-Zustandsautomaten, wie in [NAITO & TSUNOYAMA 1981] beispielhaft aufgezeigt, durch geeignete Werkzeuge auch auf GRAFCET anwenden zu können [PROVOST ET AL. 2011 A]. Allerdings ergibt sich bei der Überführung eines Grafcet in einen äquivalenten Mealy-Zustandsautomaten die Problematik der **Zustandsexplosion**, so dass bereits bei einem einfachen Grafcet die Anzahl der Zustände im entsprechenden Automatenmodell ein exponentielles Vielfaches der Anzahl der GRAFCET-Schritte betragen kann. Somit entsteht ebenfalls eine sehr umfangreiche Zahl von Testsequenzen. Ein Beispiel gibt Abbildung 5-2, in der die Struktur eines einfachen Grafcet und ein Ausschnitt des zugehörigen Erreichbarkeitsgraphen dargestellt sind. Um die Testsequenzen und damit die zu testenden Zustände des Erreichbarkeitsgraphen in ihrem Umfang einzugrenzen, fordern die Autoren in [PROVOST ET AL. 2010 B], dass sich jeweils nur ein Eingabewert von Testschritt zu Testschritt ändert. Dies stellt eine wesentliche Einschränkung für die zugrunde liegenden Grafcets und die vorgeschlagene Methode dar. Zudem wird GRAFCET nicht in vollem Umfang berücksichtigt, so dass nur solche Konstrukte verwendet werden können, die keine *zeitabhängigen Bedingungen*, *zwangsteuernden Befehle* und *einschließende Schritte* enthalten.

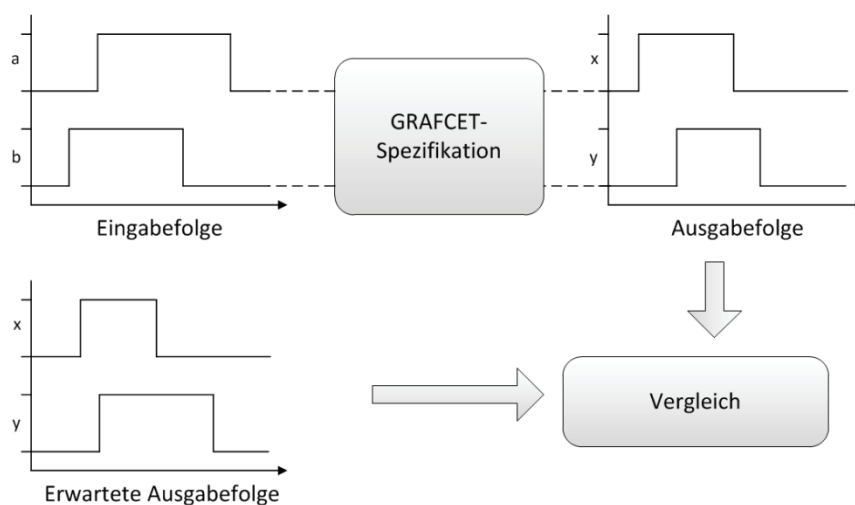


Abbildung 5-1: Prinzip der formalen Validierung durch Testsequenzen

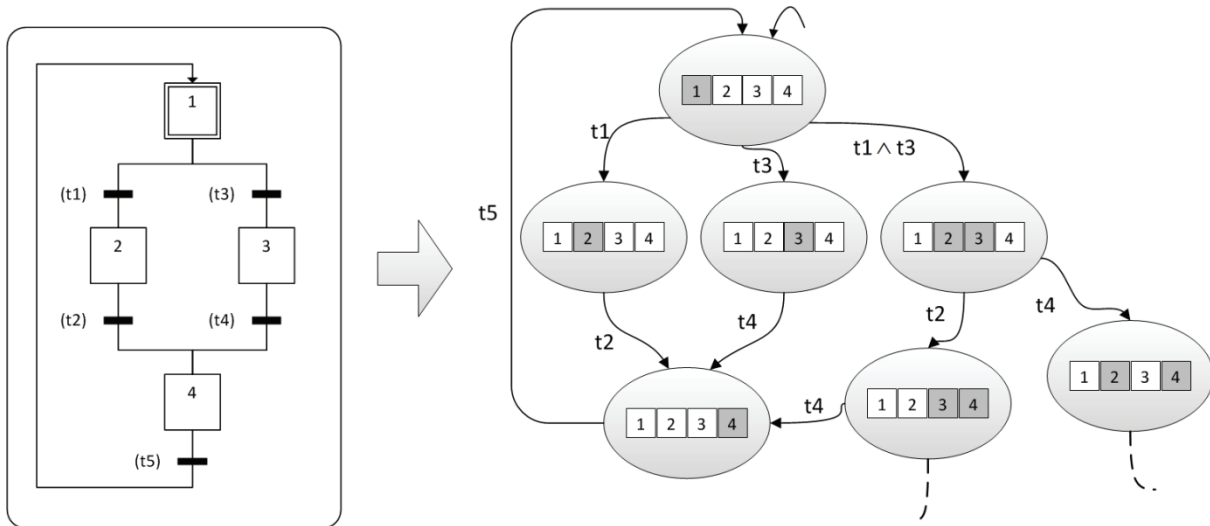


Abbildung 5-2: Ausschnitt des Erreichbarkeitsgraphen (rechts) eines einfachen Grafcet (links)

Eine abstraktere Sichtweise auf das dynamische Verhalten von GRAFCET ergibt sich durch die formale Definition in Form von endlichen Zustandsautomaten, deren Grundzüge beispielsweise in [LUNZE 2006] erörtert werden. Zustandsautomaten besitzen gegenüber Petrinetzen den Vorteil, dass weitreichende Methoden und Werkzeuge zur formalen Verifikation und Validierung verfügbar sind. Aus diesem Grund werden oftmals auch formal als SIPN definierte Grafquets durch geeignete Transformationsmethoden in Automatenmodelle überführt. In Abhängigkeit von dem Hauptaugenmerk der formalen Analyse stehen verschiedene Automatentypen zur Auswahl und entscheiden in den meisten Ansätzen über den Umfang der betrachteten Aspekte eines Grafquets. Beispielsweise können als Zustände des Automaten ausschließlich die stabilen Situationen oder sowohl stabile als auch instabile Situationen eines Grafquets in Betracht gezogen werden [L'HER ET AL. 1995]. Hieraus resultieren unterschiedliche Interpretationsalgorithmen eines Grafquets und somit auch entsprechend unterschiedliche Automatenmodelle.

In anderen Ansätzen, wie beispielsweise [FRENSEL & BRUIJN 1998], wird das Modell für die Steuerstrecke durch einen hybriden Zustandsautomaten modelliert. Das Modell der Steuerung liegt als Grafquet vor. Um nun eine gemeinsame formale Analyse von Strecken- und Steuerungsmodell durchführen zu können, wird das dynamische Verhalten von GRAFCET ebenfalls als hybrider Zustandsautomat interpretiert. Durch den Test des Steuerungsverhaltens anhand des Streckenmodells ergeben sich dann Aufschlüsse darüber, ob die Steuerungslogik den Sollablauf der Steuerstrecke garantiert.

In weiteren Forschungsansätzen, wie beispielsweise [MARCÉ ET AL. 1996, LE PARC ET AL. 1999], wird ein formales Modell des Verhaltens von GRAFCET auf Basis von synchronen Programmiersprachen vorgeschlagen, wie beispielsweise *SIGNAL* oder *Lustre*. Die zugrunde liegende Automatenstruktur ermöglicht die Anwendung der gleichen Methoden zur formalen Verifikation und Validierung. Darüber hinaus werden zeitabhängige Zusammenhänge eines Grafquets im formalen Modell berücksichtigt, was zu unendlich vielen Zuständen aufgrund des kontinuierlichen Parameters Zeit führen kann. Abhilfe hierfür wird beispielsweise in [ALUR ET AL. 1996] durch die Anwendung symbolischer Verifikationsmethoden geschaffen, indem die explizite Berücksichtigung aller erreichbaren Zustände durch die Verwendung einer temporalen Logik [HENZINGER ET AL. 1992] umgangen wird.

Um das dynamische Verhalten der Steuerung, welches durch einen Grafcet repräsentiert wird, anhand eines Modells der gesteuerten Strecke zu testen, schlagen die Autoren in [ROUSSEL & LESAGE 1996] eine alternative Möglichkeit vor. Anstatt einer expliziten Modellierung der Steuerstrecke in Form eines bestimmten Automatentyps können diese auch indirekt durch Bedingungen formuliert werden, die einer definierten Grammatik unterliegen. Im Rahmen der Validierung werden diese Bedingungen dann auf das formale Modell der Steuerung angewendet.

Sowohl die Ansätze unter Verwendung von Zustandsautomaten als auch diejenigen unter Verwendung synchroner Programmiersprachen stehen vor dem Problem der exponentiellen Zunahme der Zustände im formalen Modell für die Abbildung eines GRAFCET-äquivalenten Verhaltens. Dies führt zu teilweise sehr zeitaufwändigen Berechnungsverfahren. Zusätzlich berücksichtigen alle vorgestellten Ansätze nur eine Teilmenge der in IEC 60848 definierten Elemente für GRAFCET.

5.1.2 Ansätze zur Steuerungssynthese

Die Steuerungssynthese befasst sich mit der (teil-) automatischen Generierung des Steuerungsmodells auf der Grundlage eines formalen Modells für das Verhalten des ungesteuerten Prozesses bzw. der Anlage und formal definierter Anforderungen an das Automatisierungssystem (→ 2.2.3). Auch auf diesem Gebiet der formalen Methoden gibt es Ansätze, welche das Beschreibungsmittel GRAFCET verwenden, um die Anforderungen an das Automatisierungssystem zu spezifizieren. Da diese Ansätze in Anlehnung an die theoretischen Grundlagen zur Zustandsüberwachung in ereignisdiskreten Systemen gemäß [RAMADGE & WONHAM 1986] entstanden sind, nach denen das resultierende formale Modell der überwachenden Steuerung ein endlicher Zustandsautomat ist, wird GRAFCET dazu verwendet, um benötigte Eingangsinformationen der Syntheselgorithmen eindeutig zu beschreiben. Wie bereits in Bezug auf die Methoden zur formalen Verifikation und Validierung erwähnt, ist auch für die Steuerungssynthese eine formale Definition für GRAFCET unabdingbar. Einen umfassenden Ansatz für die Steuerungssynthese mit GRAFCET vermittelt [ZAYTOON 2002].

Ein erster Ansatz zur Steuerungssynthese mit GRAFCET wird in [CHARBONNIER ET AL. 1999] veröffentlicht, wobei dessen Vorzüge anhand einer realen Steuerstrecke demonstriert werden. Jedoch ist GRAFCET in diesem Fall einigen Einschränkungen unterworfen, so dass nur elementare GRAFCET-Elemente verwendet werden. So dürfen bezüglich des Wirkungsteils ausschließlich einfache Ereignisse in Transitionsbedingungen spezifiziert und nur *gespeichert wirkende Aktionen* mit Schritten assoziiert werden. Die Ausführung einer *gespeichert wirkenden Aktion* ist hier unmittelbar mit einem Zustandsübergang der Steuerstrecke verbunden, so dass gleichzeitige Änderungen nicht möglich sind. Aufgrund der starken Restriktionen für GRAFCET in [CHARBONNIER ET AL. 1999] wird in [ZAYTOON & CARRÉ-MÉNÉTRIER 2001] eine erweiterte methodische Vorgehensweise zur Steuerungssynthese vorgeschlagen (→ Abbildung 5-3). Das resultierende formale Modell der Steuerung erfüllt zwar diverse Anforderungen, wie beispielsweise ein deterministisches Verhalten oder die explizite Vermeidung sicherheitskritischer, verbotener Zustände. Eine vollständige Berücksichtigung von GRAFCET unter Einbeziehung aller in IEC 60848 definierten Elemente erfolgt aber auch in diesem Falle nicht. GRAFCET beschreibt in diesem Fall, entgegen dem

Verständnis dieser Arbeit, das Verhalten der gesteuerten Strecke, also deren Sollverhalten. Ausgehend von dem Erreichbarkeitsgraphen des Grafcet wird dessen Zustandsraum durch die Berücksichtigung von Anforderungen an die Sicherheit der Steuerung eingeschränkt. Er repräsentiert somit die formalisierten Anforderungen an die Steuerung. Werden diese mit einem Modell für das Verhalten der ungesteuerten Steuerstrecke verknüpft, so kann daraus das korrekte Modell der Steuerung algorithmengestützt abgeleitet und implementiert werden. Die Autoren räumen allerdings ein, dass die von ihnen erörterte Methode stark von der Qualität des Streckenmodells abhängig ist und schlagen eine iterative Vorgehensweise in Abhängigkeit von den vorhandenen Streckenmodellen vor. Eine Fortsetzung dieser Methode und detailliertere Ausführungen zu den einzelnen Vorgehensschritten aus Abbildung 5-3, wie beispielsweise die Formalisierung von sicherheitsgerichteten Anforderungen, finden sich in [CARRÉ-MÉNÉTRIER & ZAYTOON 2002]. In [CARRÉ-MÉNÉTRIER ET AL. 1999] wird in diesem Zusammenhang ein Werkzeug präsentiert, welches die Anwendung der Steuerungssynthese nach den methodischen Vorgaben und unter Verwendung von GRAFCET unterstützt.

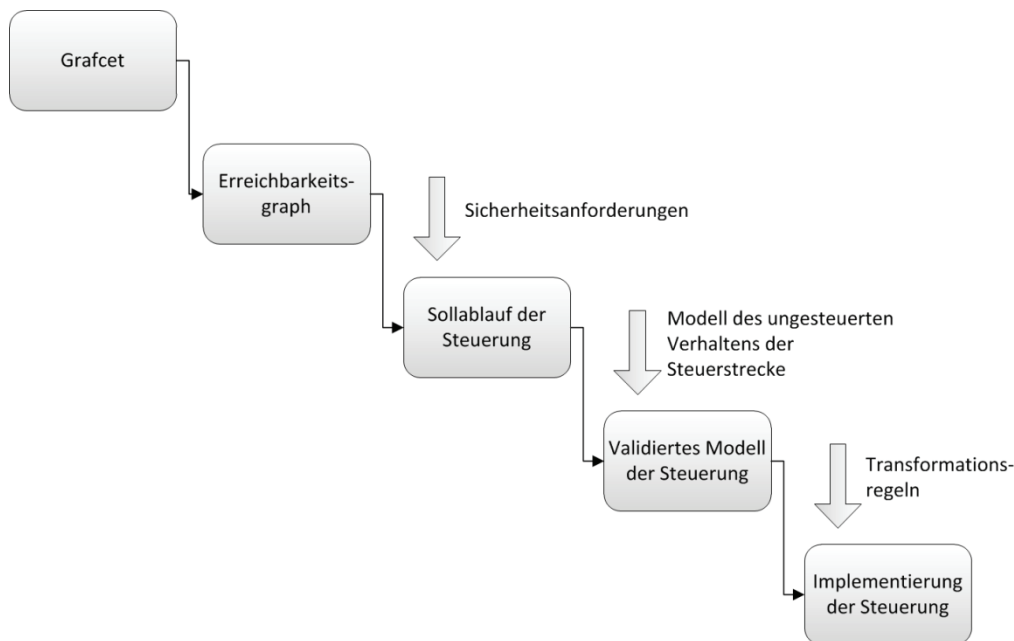


Abbildung 5-3: Methode zur Steuerungssynthese, in Anlehnung an [ZAYTOON & CARRÉ-MÉNÉTRIER 2001]

Zwar bietet die in Abbildung 5-3 gezeigte grundsätzliche Vorgehensweise bei der Synthese von Steuerungsmodellen mit GRAFCET ein einfaches Konzept. Wesentliche Schwierigkeiten ergeben sich aber in Bezug auf eine geeignete formale Beschreibung von einschränkenden Bedingungen sowie einer geeigneten Modellierung des ungesteuerten Verhaltens der Steuerstrecke. Zudem bleibt bei den auf Erreichbarkeitsgraphen basierenden Methoden nach wie vor die Problematik der *Zustandsexplosion*. Diesbezüglich wird in [PHILIPPOT & TAJER 2010] ein Algorithmus für die Generierung eines modifizierten und auf eine vereinfachte Steuerungssynthese zugeschnittenen Erreichbarkeitsgraphen vorgeschlagen. Somit kann die hohe Anzahl an theoretisch möglichen Zuständen reduziert werden.

5.1.3 Ansätze zur automatischen Generierung

Auch im Bereich der automatischen Generierung gibt es einige Forschungsansätze, die GRAFCET als Ausgangspunkt für eine rechnergestützte Transformation in lauffähigen Steuerungscode integrieren. Idealerweise dient der jeweilige Grafcet dann zusätzlich als eine

übersichtliche Dokumentation des Steuerungsablaufs, besonders wenn dieser in einer textuellen Programmiersprache implementiert wird [THOMAS & MCLEAN 1988]. Allen Forschungsansätzen gemeinsam ist, dass GRAFCET stets als Ausgangspunkt für die automatische Generierung betrachtet wird. Eine methodische Vorgehensweise, wie aus vorhandenen Steuerungsprogrammen entsprechende Grafquets generiert werden können, existiert nach derzeitigem Kenntnisstand nicht. Insbesondere in diesem Fall der automatischen Generierung (Re-Engineering → 2.2.1) wäre eine möglichst vollständige formale Definition von GRAFCET hilfreich, um das dynamische Verhalten vorhandener Steuerungsprogramme eindeutig in einen detaillierten und übersichtlichen Grafquet zu überführen.

Zur Unterstützung eines strukturierten Vorgehens bei der Implementierung von Steuerungsalgorithmen schlagen [GAERTNER & THIRION 1999] eine auf Softwaremustern (*patterns*) basierende Methode vor. GRAFCET wird in diesem Ansatz dazu verwendet, die Sequenz des Steuerungsalgorithmus grafisch zu beschreiben und zu dokumentieren. Die Implementierung erfolgt anschließend manuell, mittels objektorientierter Programmierung, auf der Grundlage der als *Grafquet-Muster* bezeichneten Spezifikation. Eine automatische Generierung wird zunächst nicht diskutiert. Die auf den *Grafquet-Mustern* basierende Methode fokussiert hingegen auf den Aspekt der Wiederverwendung im Falle ähnlicher oder gleicher Steuerungsaufgaben.

Einen Compiler-basierten Ansatz zur automatischen Generierung von Code auf der Grundlage eines Grafquets schlagen [MALLET ET AL. 2000] vor. Um einen Grafquet zu transformieren, diskutieren die Autoren zwei unterschiedliche Ansätze zur Implementierung des GRAFCET-Verhaltens in der Hardware-Beschreibungssprache *Very High Speed Integrated Circuit Hardware Description Language* (VHDL) [IEEE 1076][#]. Ein Ansatz orientiert sich an den fünf Ablaufregeln von GRAFCET, ein weiterer implementiert das dynamische Verhalten ausgehend von dem zugehörigen Erreichbarkeitsgraphen. Beide Ansätze berücksichtigen die Definitionen der IEC 60848 aber nur unvollständig, so dass im Wesentlichen die durch die Struktur vorgegebene Schrittkette den Ausgangspunkt für die VHDL-Transformation bildet. Wirkungsteil, *zeitabhängige Bedingungen* und Elemente zur hierarchischen Strukturierung werden nicht berücksichtigt.

Die Anwendung von Methoden zur grafischen Programmierung von Steuerungsalgorithmen im Bereich der Mikrocontroller-Programmierung für eingebettete Systeme wird in [BAYÓ-PUXAN ET AL. 2008] aufgezeigt. Die grafische Spezifikation der Steuerungsalgorithmen durch GRAFCET wird hierbei als eine geeignete übersichtliche Methode angesehen, um den Zugang zu Mikrocontroller-Programmen zu erleichtern, welche in der Regel mit prozeduralen Programmiersprachen erstellt werden. Dazu diskutieren die Autoren zunächst, wie das durch GRAFCET spezifizierte dynamische Verhalten für die automatische Generierung von C-Code interpretiert werden muss, damit das Verhalten der Implementierung der Spezifikation entspricht. Neben den grundlegenden GRAFCET-Elementen der Schritte, Transitionen und Wirkverbindungen werden in diesem Ansatz auch Ereignisse sowie *gespeicherte* und *kontinuierlich wirkende Aktionen* gemäß IEC 60848 berücksichtigt.

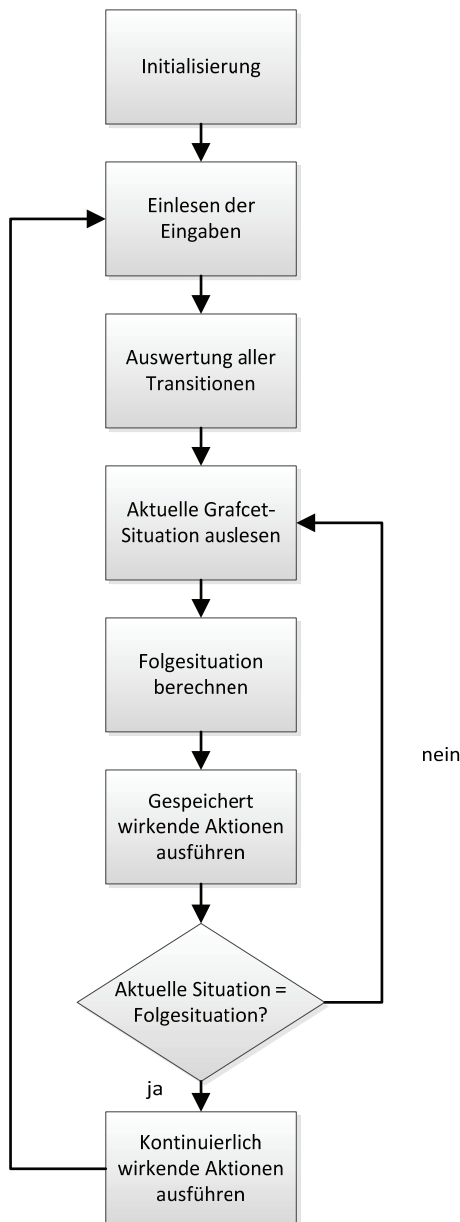


Abbildung 5-4: Programmablauf für die Implementierung von GRAFCET, in Anlehnung an [BAYÓ-PUXAN ET AL. 2008]

Die bereits in [L'HER ET AL. 1995] erörterte Problematik der Reihenfolge zur Auswertung und Ausführung von Transitionsbedingungen, Transitionen und Aktionen bei der Implementierung eines Grafcet auf Ein-Prozessorsystemen führt zu dem Vorschlag eines sequentiellen Programmablaufs (→ Abbildung 5-4). Die vorrangige Evaluation der freigegebenen Transitionen und ihrer Transitionsbedingungen führt allerdings dazu, dass möglicherweise mit Initialschritten assoziierte *kontinuierlich wirkende Aktionen* keine Berücksichtigung im Programmablauf finden. Eine vorrangige Ausführung *gespeichert wirkender* und *kontinuierlich wirkender Aktionen* könnte diesbezüglich Abhilfe schaffen, wurde aber in [BAYÓ-PUXAN ET AL. 2008] nicht erwähnt.

Ein auf GEMMA basierender Ansatz für eine systematische Vorgehensweise zur automatischen Generierung IEC 61131-3 konformer Steuerungs-algorithmen wird in [ALVAREZ ET AL. 2012] vorgestellt. Der Schwerpunkt liegt auf der Entwicklung eines Werkzeugs, welches den Anwender durch die einzelnen Entwurfs- und Spezifikationsschritte führt und als Ausgabe *PLCopenXML* konformen Steuerungscode automatisch generiert. Durch die Nutzung der *PLCopenXML* Schnittstelle ist abschließend ein Import in *Multiprog* vorgesehen. Die Werkzeugunterstützung für GRAFCET erfolgt durch *SFCedit* (→ 4.5). Die systematische Vorgehensweise sieht vor, dass nach der Erstellung des GEMMA-Graphen, welcher die Betriebsarten des Steuerungssystems in Form von Zuständen und die

Bedingungen für den Betriebsartenwechsel in Form von Zustandsübergängen grafisch dokumentiert, die Steuerungsabläufe innerhalb der jeweiligen Betriebsart durch Grafkets spezifiziert werden. Aus dem Grafcet für den jeweils grundlegenden Steuerungsablauf heraus können über *kontinuierlich wirkende Aktionen* untergeordnete Sequenzen aufgerufen werden (→ Abbildung 5-5), so dass eine eingebettete GRAFCET-Struktur entsteht, die den Vorgaben für SFCs entspricht. Im Zuge der Transformation in einen IEC 61131-3 konformen Steuerungsalgorithmus resultiert aus der eingebetteten GRAFCET-Struktur ein SFC-Programm, aus dem entsprechende Funktionsbausteine über Aktionen aufgerufen werden (→ Abbildung 5-6).

Werkzeugseitig werden die in *SFCedit* erstellten Grafjets in dem zugehörigen proprietären XML-Format von *SFCedit* abgespeichert und durch die Adaption eines in [LÜDER ET AL. 2010] vorgestellten softwarebasierten Transformationsgerüsts in eine *PLCopenXML*-Datei überführt. Um eine softwaregestützte Transformation des *SFCedit*-XML-Schemas in das *PLCopenXML*-Schema zu etablieren, wurde aufgrund von beispielhaft erstellten Modellen in *SFCedit* die zugehörige XML-Struktur analysiert und ein XML-Schema extrahiert. Insgesamt betrachtet, ermöglicht diese Methode ein systematisches Vorgehen im Steuerungsentwurf. Die Sichtweise bezüglich GRAFCET ist allerdings sehr stark an SFCs ausgerichtet, was einerseits die Durchgängigkeit der Methode ermöglicht, aber andererseits die Mächtigkeit des Beschreibungsmittels GRAFCET deutlich einschränkt. So ist gemäß den Definitionen in [DIN EN 60848][#] eine Einbettung von untergeordneten Grafjets in einen übergeordneten Grafjet über *kontinuierlich wirkende Aktionen* nicht vorgesehen, wohingegen dies ein zulässiges Konstrukt von SFCs darstellt. Weitere Forschungsarbeiten im Bereich der automatischen Generierung präsentieren Ansätze für eine grafische Programmierung von Industrierobotern durch GRAFCET, wie beispielsweise die Beiträge [MARTINEZ & MARTINEZ 2005] oder [ARNOLD & HENRIQUES 2005]. Programmabläufe für die vorgesehenen Robotersequenzen werden grafisch durch eine GRAFCET-ähnliche Struktur programmiert. Die Autoren in [ARNOLD & HENRIQUES 2005] stellen diesbezüglich ein geeignetes Werkzeug vor. GRAFCET wird somit vorrangig als Programmiersprache angesehen, was der im Rahmen dieser Arbeit verfolgten Sichtweise von GRAFCET als einer technologieunabhängigen Spezifikationsprache für Steuerungsabläufe widerspricht.

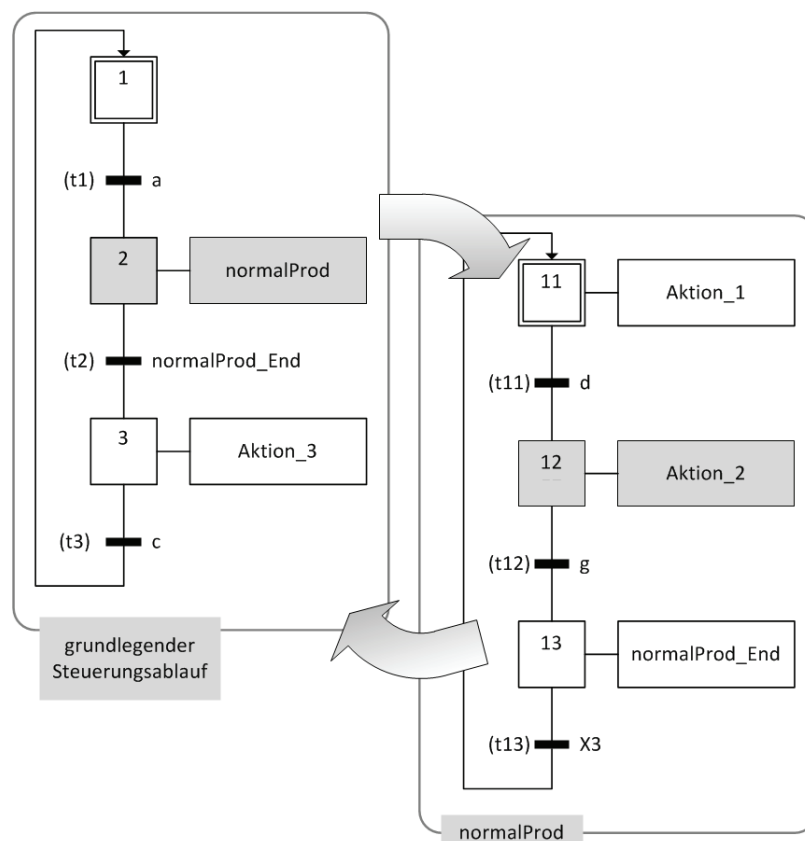


Abbildung 5-5: Hierarchische GRAFCET-Struktur nach [ALVAREZ ET AL. 2012]

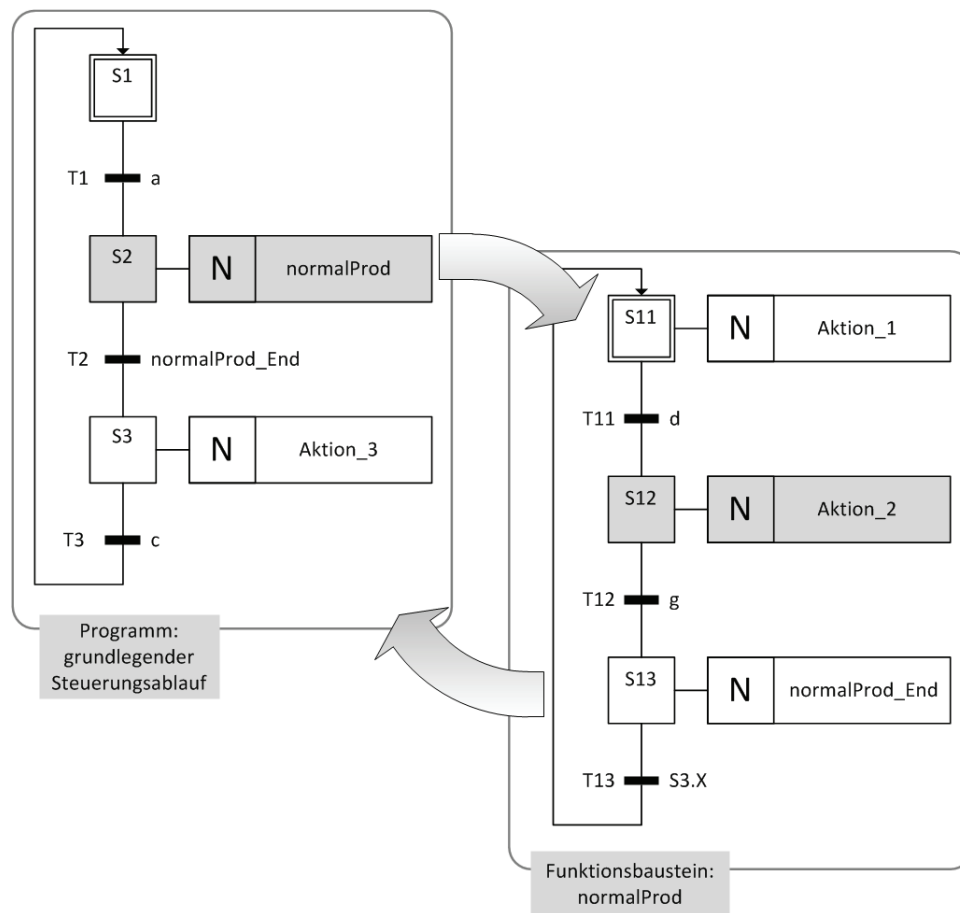


Abbildung 5-6: Direkte Abbildbarkeit der GRAFCET-Struktur aus Abbildung 5-5 in eine Programmstruktur nach IEC 61131-3

5.2 Handlungsbedarf bezüglich GRAFCET

GRAFCET lässt sich in die Kategorie der semiformalen Beschreibungsmittel einordnen, so dass als grundlegende Voraussetzung für die Anwendung formaler Methoden die Überführung in ein formales Modell notwendig ist. Allen bisherigen Forschungsansätzen gemeinsam ist die Frage nach einer eindeutigen formalen, mathematischen Notation für die Struktur und das dynamische Verhalten von GRAFCET. Es werden im Wesentlichen zwei Ansätze diskutiert, die auf bereits bestehenden, mathematisch eindeutig definierten Beschreibungsmitteln beruhen. Sie beschreiben einerseits die Definition von GRAFCET als SIPN, andererseits die Definition des GRAFCET-Verhaltens als endlichen Zustandsautomaten. Im Zusammenhang mit der Wahl des formalen Modells müssen stets der Verwendungszweck sowie die Frage, welchen Sachverhalt GRAFCET im jeweiligen Forschungsansatz spezifiziert, beachtet werden. Petrinetze, insbesondere SIPN, vermögen die Struktur und das dynamische Verhalten von GRAFCET in seiner ursprünglichen Art und Weise abzubilden, wohingegen Zustandsautomaten eine abstraktere, übergreifende Sichtweise auf das dynamische Verhalten eines Grafcet zulassen. Insgesamt liegen die Schwerpunkte der derzeitigen Forschungsansätze für einen werkzeugunterstützten Steuerungsentwurf mit GRAFCET auf der formalen Verifikation und Validierung, der Steuerungssynthese sowie der semantischen Integration weiterführender Sprachelemente [LHOSTE ET AL. 1993]. Allerdings werden sowohl Struktur, als auch dynamisches Verhalten von GRAFCET gemäß den Definitionen der IEC 60848 bisher nicht vollständig betrachtet. Wesentliche Vorteile von

GRAFCET, wie die Möglichkeit, zeitabhängige Anforderungen zu spezifizieren oder einen Grafcet durch hierarchische Konstrukte modular in miteinander verknüpfte Teil-Grafcets zu gliedern, können somit nicht genutzt und in die formale Analyse integriert werden. Tabelle 5-1 zeigt hierzu eine Übersicht über den Stand der Forschung bezüglich GRAFCET. Für die einzelnen Forschungsansätze wird zusammenfassend dargestellt, welches Ziel jeweils mit GRAFCET verfolgt wird und welche grafischen Elemente gemäß IEC 60848 berücksichtigt werden.

Weiterer Handlungsbedarf besteht hinsichtlich allgemein akzeptierter Regeln für den Entwurf von Grafcets. Die in den vorangegangenen Kapiteln diskutierten Forschungsarbeiten bezüglich formaler Methoden mit GRAFCET schlagen zwar eine systematische Vorgehensweise und algorithmische Umsetzung, beispielsweise für die Generierung von Steuerungsalgorithmen, vor, geben aber keine konkreten Modellierungsregeln für die Erstellung eines Grafcet. Einige Modellierungsregeln sind in IEC 60848 bereits enthalten:

„Der Entwickler muss sicherstellen, dass die Transitionsbedingungen vom zeitlichen, logischen oder mechanischen Aspekt her untereinander exklusiv sind.“
[DIN EN 60848, S.29][#].

Ein zusammenhängendes, allgemein akzeptiertes und vollständiges Regelwerk existiert allerdings nicht. Am Beispiel eines Chargenprozesses diskutiert der Beitrag [FERRARINI 2006] eine konkrete, geeignete Richtlinie für eine modulare Spezifikation und Implementierung eines Steuerungsablaufs. Weitere Anwendungsbeispiele wären wünschenswert, um die Vorzüge von GRAFCET zu unterstreichen.

Die IEC 60848 definiert die Zusammenhänge bezüglich GRAFCET in textueller Form, die rechnergestützte Handhabung eines Grafcet liegt allerdings außerhalb des Betrachtungshorizontes dieser Norm. Die fehlende Einbettung in eine rechnergestützte Engineering-Werkzeuglandschaft lässt GRAFCET für den betroffenen Anwender zwar als ausdrucksstarkes Beschreibungsmittel erscheinen, welches aber zusätzliche Arbeitsschritte im Steuerungsentwurf erfordert. Zweck und Vorteil einer aussagekräftigen Spezifikation in GRAFCET könnten mehr Anklang auf industrieller Ebene finden, wenn der Übergang von der Spezifikation hin zur Implementierung (teil-) automatisiert erfolgen könnte, zum Beispiel indem ein Grafcet softwaregestützt in eine Implementierung nach IEC 61131-3 übersetzt würde. Die DIN EN 60848 gibt diesbezüglich folgenden Hinweis im Anhang C:

„Ein zukünftiger integrierter Ansatz wurde vom IEC/SC 3B und IEC/TC 65 diskutiert, um einen textuellen Ausgang von IEC 60848 zu ermöglichen, der automatisch in eine Struktur und ein Programm nach IEC 61131-3 bzw. IEC 61499 umgesetzt werden kann.“ [DIN EN 60848, S. 49][#].

Allerdings ist ein solcher Ansatz mit heutigem Kenntnisstand weder realisiert noch festgelegt worden und bedarf weiterer Forschungsarbeit. Ein Vergleich der IEC-Standards von 2002 [IEC 60848 A][#] und 2013 [IEC 60848 B][#] zeigt, dass Anhang C im Rahmen der Überarbeitung nicht verändert wurde.

Auf der Grundlage eines Grafcet existieren Ansätze für Methoden und Werkzeuge zur automatischen Generierung von Steuerungscode (→ 5.1.3). Allerdings existiert nach derzeitigem Kenntnisstand keine Methode zur automatischen Generierung IEC 61131-3 konformen

Steuerungscode, welcher im Bereich des Engineerings der Automatisierungstechnik weit verbreitet und in der praktischen Anwendung akzeptiert ist. Ein Werkzeug, welches diese Methode unterstützt sowie die dafür notwendigen Transformationsregeln implementiert und die durch den Grafcet dokumentierten Informationen bezüglich des Steuerungsablaufs in einem offenen Dateiformat zugänglich macht, ist zurzeit nicht verfügbar.

Tabelle 5-1: Übersicht über den Stand der Forschung

Veröffentlichung	Ziel				Abdeckung						
	Verifikation	Validierung durch Test	Synthese GRAFCET beschreibt Sollverhalten des Systems	Codegenerierung GRAFCET beschreibt Steuerungsablauf	Basic-Grafcet ohne Aktionen	kont. wirkende Aktionen	gesp. wirkende Aktionen	zeitabhängige Bedingungen	Makroschritte	einschließende Schritte	zwangsteuernde Befehle
[ALVAREZ ET AL. 2012]				X	X	X	X				
[ARNOLD & HENRIQUES 2005]				X	X	X	X				
[BAYO-PUXAN ET AL. 2008]				X	X	X	X				X
[CARRÉ-MÉNÉTRIÉR & ZAYTOON 2002]			X		X	X	X				
[CHARBONNIER ET AL. 1999]			X		X	X	X				
[CHÉRIAUX ET AL. 2010]		X			X	X	X		X		
[EL RHALIBI ET AL. 1995]	X				X						
[FRENSEL & BRUIN 1998]	X				X	X					
[GAERTNER & THIRION 1999]				X	X	X					
[L'HER ET AL. 1995]	X				X			X			
[LE PARC ET AL. 1999]	X				X	X	X		X		
[MALLET ET AL. 2000]				X	X	X					
[MARCÉ ET AL. 1996]	X				X			X			
[MARTINEZ & MARTINEZ 2005]				X	X	X					
[PHILIPPOT & TAÏER 2010]			X		X	X					
[PROVOST ET AL. 2009]					X	X	X		X		
[PROVOST ET AL. 2010 A]		X			X	X	X		X		
[PROVOST ET AL. 2010 B]					X	X	X		X		
[PROVOST ET AL. 2011 B]					X	X	X		X		
[ROUSSEL 1994]	X				X	X	X				X
[ROUSSEL & LESAGE 1996]	X	X			X	X	X				
[ZAYTOON & CARRÉ-MÉNÉTRIÉR 2001]			X		X	X					

Legende: X: wird abgedeckt / berücksichtigt

Der häufigste Anwendungsfall im Steuerungsentwurf des Engineerings automatisierter Anlagen ist dadurch gekennzeichnet, dass Steuerungsprogramme ausgehend von einer in der Planungsphase erstellten, dokumentenbasierten Spezifikation der funktionalen Anforderungen an den Steuerungsalgorithmus (Steuerungsablauf) direkt implementiert werden. Bezogen auf den Steuerungsentwurf, liegt der Schwerpunkt auf einer vorwärtsgerichteten Vorgehensweise (→ 2.1). Ein Beschreibungsmittel, welches die wesentlichen Zusammenhänge des Steuerungsablaufs grafisch veranschaulicht, wie beispielsweise GRAFCET, könnte insbesondere an der Gewerke-Schnittstelle zwischen Planung und Realisierung zu einem gemeinsamen eindeutigen Verständnis beitragen. Ein Grafcet muss momentan aber immer noch manuell durch den Anwender in ein entsprechendes Steuerungsprogramm überführt werden. Bezüglich der Planung und Realisierung automatisierter Anlagen ergäbe sich durch eine solche eingeschränkte Nutzung von GRAFCET zusätzlicher, nicht zu rechtfertigender Aufwand, einerseits für die Erstellung des Grafkets in der Planungsphase und andererseits für die manuelle Umsetzung und Pflege des Grafkets in ein lauffähiges Steuerungsprogramm während der Inbetriebnahme und der Betriebsphase. Formale Methoden erzielen an dieser Stelle eine große Hebelwirkung für ein effizienteres Engineering, indem sie durch eine automatische Generierung von Steuerungscode eine rechnergestützte Interpretation und Handhabung der Spezifikation des Steuerungsablaufs ermöglichen. Die mathematisch eindeutige Definition von Struktur und dynamischem Verhalten (formales Modell) des grafischen Beschreibungsmittels ist hierfür eine notwendige Bedingung.

Der im Rahmen dieser Arbeit vorgestellte Ansatz fokussiert aus diesem Grund auf den Steuerungsentwurf während der Planung und Realisierung automatisierter Anlagen und die automatische Generierung IEC 61131-3 konformen Steuerungscode, ausgehend von einer Spezifikation des Steuerungsablaufs mit Hilfe des Beschreibungsmittels GRAFCET. Abbildung 5-7 verdeutlicht hierzu den in den nachfolgenden Kapiteln 6 bis 8 dargestellten Beitrag dieser Arbeit. Die grundlegende Zielsetzung besteht darin, einen methodischen Ansatz für den Weg von einer Steuerungsspezifikation in GRAFCET gemäß IEC 60848 zu IEC 61131-3 konformem Steuerungscode mit Hilfe formaler Methoden zur automatischen Generierung aufzuzeigen und diesen gezielt durch softwarebasierte Werkzeuge zu unterstützen. In einem ersten Schritt zur Formalisierung wird die auf SIPN basierende formale Definition für GRAFCET in [DAVID & ALLA 2010] so erweitert, dass auch *einschließende Schritte*, *zwangsteuernde Befehle* und *zeitabhängige Bedingungen* Berücksichtigung finden. Somit werden alle bedeutenden GRAFCET-Elemente gemäß der Definitionen der IEC 60848 auf eine mathematisch eindeutige Grundlage gestellt und die Anwendung formaler Methoden grundsätzlich möglich. Die zugehörigen Definitionen zu einem formalen SIPN-Modell für GRAFCET und dessen Erweiterungen werden in Kapitel 6 erläutert.

Das anschließende Kapitel 7 beinhaltet einen Vorschlag für eine implementierungsunabhängige Notation für GRAFCET und bildet den zweiten Schritt im Rahmen der Formalisierung. Im Hinblick auf eine durch softwarebasierte Werkzeuge unterstützte automatische Generierung sollten die Inhalte eines Grafcet in einem offenen Datenaustauschformat abgelegt werden können, so dass andere Werkzeuge einfach auf die gespeicherten Daten zugreifen können. Die GRAFCET-spezifischen Anforderungen an ein entsprechendes offenes Format werden etablierten XML-Notationen gegenübergestellt mit dem Ziel, ein bereits vorhandenes XML-Schema als Datenaustauschformat nutzen zu können. Das Konzept und die

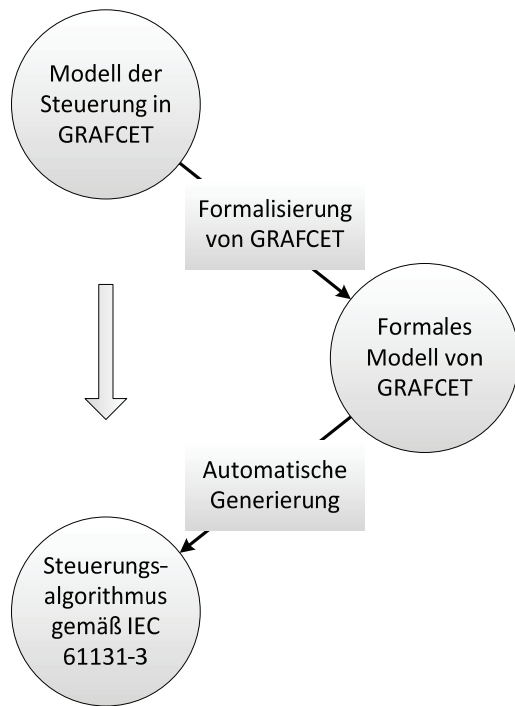


Abbildung 5-7: Übersicht über die systematische Vorgehensweise in dieser Arbeit

Umsetzung einer implementierungsunabhängigen Notation für GRAFCET werden in der zweiten Hälfte des Kapitels 7 dargestellt. Ausgehend von dem formalen Modell von GRAFCET können Abbildungsregeln definiert werden, welche die Transformation eines Grafnet in IEC 61131-3 konformen Steuerungscode ermöglichen. Anforderungen, Konzeption und Umsetzung der Abbildungsregeln werden in Kapitel 8 erörtert, wobei die Ähnlichkeit und enge Verwandtschaft von GRAFCET mit SFCs einen entscheidenden Aspekt darstellt. Die Abbildungsregeln bilden den Kern der im Rahmen dieser Arbeit vorgestellten Methode zur automatischen Generierung.

Wie bereits in Kapitel 4.5 beschrieben, ist ein wesentliches Merkmal momentan verfügbarer Werkzeuge für die Erstellung von Grafnetts die fehlende Offenheit des Datenformats. Die

Informationen eines Grafnetts liegen in einem proprietären Datenformat vor, welches für eine werkzeuggestützte Weiterverarbeitung ungeeignet ist. Die in Kapitel 9 aufgezeigte prototypische Implementierung der zuvor beschriebenen formalen Zusammenhänge umfasst aus diesem Grund sowohl ein Werkzeug für die Erstellung von Grafnetts (**GRAFCET-Editor**), dessen XML-basiertes Datenformat allgemein zugänglich ist und in ein werkzeugunabhängiges Datenformat überführt werden kann, als auch ein Werkzeug für die automatische Generierung IEC 61131-3 konformen Steuerungscode (**GRAFCET-Transformator**).

6 Mathematische Basis für das Beschreibungsmittel GRAFCET

6.1 Grundlegende formale Definition von GRAFCET als SIPN

6.1.1 Das formale Modell eines Basic-Grafcet

Die Diskussion des bezüglich GRAFCET veröffentlichten Standes der Forschung (→ 5.1) zeigt grundsätzlich zwei Ansätze zu einer formalen Definition. Bei der formalen Definition als Zustandsautomat steht das dynamische Verhalten eines Grafcet im Vordergrund, welches im Wesentlichen durch sein Ausgabeverhalten charakterisiert ist. Zustände und Zustandsübergänge des Zustandsautomaten spiegeln das externe, nach außen wahrnehmbare Verhalten eines Grafcet wider. Interne Ereignisse und die Struktur werden durch den Abstraktionsgrad dieses formalen Modells hingegen vernachlässigt.

Eine detaillierte Perspektive ergibt sich bei der formalen Definition als steuerungstechnisch interpretiertes Petrinetz. Aufgrund der historisch begründeten Abstammung von SIPN bieten deren mathematische Grundlagen, ergänzt durch die Definitionen der *Algebra of Events* (→ 4.2.2), eine formale Basis für GRAFCET. In [DAVID & ALLA 2010, S.357] werden daraus zwei zentrale Thesen abgeleitet, die hinsichtlich einer formalen Definition für GRAFCET als SIPN gelten:

These 1:

Wird ein SIPN als Grafcet interpretiert, so hat es ein exakt gleiches Verhalten.

These 2:

Wird ein Grafcet als SIPN interpretiert, so hat er ein exakt gleiches Verhalten.

These 1 besagt, dass ein mit Hilfe von SIPN spezifiziertes Steuerungsverhalten direkt in einen Grafcet mit äquivalentem Verhalten überführt werden kann. Das SIPN muss dazu eine endliche Anzahl erreichbarer Zustände besitzen, in denen die einzelnen Stellen stets nur eine Marke beinhalten, synchron zu externen Ereignissen und deterministisch sind. Somit kann der grundsätzliche Unterschied einer binären Schrittmarkierung in GRAFCET und einer numerischen Stellenmarkierung in SIPN überwunden werden. Umgekehrt ist es möglich, einen Grafcet direkt in ein äquivalentes SIPN zu überführen, wie in These 2 formuliert. Die Beschreibungsmächtigkeit von GRAFCET ist der von SIPN also zumindest gleichwertig. Dieser Sachverhalt stellt die wesentliche Motivation dafür dar, GRAFCET formal als SIPN zu definieren. Die direkte Überführung eines Grafcet in ein SIPN unterliegt allerdings gewissen Rahmenbedingungen, die in [DAVID & ALLA 2010] postuliert werden und auch für die nachfolgenden Betrachtungen im Rahmen dieser Arbeit gültig sind.

So muss es sich bei der betrachteten Spezifikation um einen gemäß den Definitionen in IEC 60848 [DIN EN 60848][#] gültigen Grafcet handeln. Dies bedeutet, dass nur die gemäß der Norm zulässigen Elemente und Konstrukte verwendet werden sollen. Ein gültiger Grafcet spezifiziert dann das Verhalten der Steuerungseinheit mit den entsprechenden Schnittstellen (→ Abbildung 4-1). Eine weitere notwendige Bedingung ist die Wohlgeformtheit des Grafcet (*sound Grafcet*, [DAVID & ALLA 2010, S.362]).

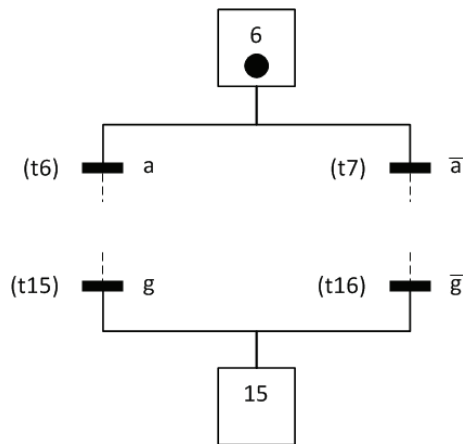


Abbildung 6-1: Beispiele für sich gegenseitig ausschließende Transitionsbedingungen

In einem wohlgeformten Grafcet kann ein in einer Situation aktiver Schritt in einer nachfolgenden Situation nicht erneut aktiviert werden, es sei denn, er wird gleichzeitig deaktiviert. Die steigende Flanke einer Schrittvariablen, wie beispielsweise ($\uparrow X_6$) in Abbildung 6-1, kann also nur dann auftreten, wenn der Schritt zuvor inaktiv war oder im Zuge eines Zustandsübergangs kurzzeitig deaktiviert wurde. Diese Forderung ist eine Konsequenz aus Ablaufregel 5 (\rightarrow 4.2.3). Besitzen zwei oder mehr Transitionen denselben Vorgängerschritt, wie im Fall von alternativen Verzweigungen, so müssen sich die zugehörigen Transitionsbedingungen gegenseitig ausschließen.

Dies bedeutet, dass die Transitionen t_6 und t_7 im Grafcet aus Abbildung 6-1 nicht gleichzeitig auslösen dürfen. Eine solche alternative Verzweigung sollte als *XOR-Verknüpfung* verstanden und entsprechend in den Transitionsbedingungen berücksichtigt werden. Ein simultanes Auslösen von Transitionen ist in einem Grafcet gemäß Ablaufregel 4 möglich, kann aber zu unerwartetem nebenläufigem Verhalten führen. Im Umkehrschluss muss für einen wohlgeformten Grafcet sichergestellt sein, dass ein Schritt nicht gleichzeitig von zwei oder mehr Transitionen aktiviert werden kann. Dies kann beispielsweise dadurch erreicht werden, dass sich die zugehörigen Transitionsbedingungen gegenseitig ausschließen, wie für die Transitionen t_{15} und t_{16} in Abbildung 6-1 dargestellt, oder auf andere Weise sichergestellt ist, dass t_{15} und t_{16} nie gleichzeitig auslösen können. Des Weiteren müssen in einem wohlgeformten Grafcet Quelltransitionen [DIN EN 60848, S.33][#] mit einem Ereignis verknüpft sein.

Ein demnach gültiger und wohlgeformter Grafcet wird in [DAVID & ALLA 2010] als Basic-Grafcet bezeichnet und verhält sich, unter Berücksichtigung der *Algebra of Events*, exakt gleich zu einem SIPN. Ein Basic-Grafcet besteht aus den Elementen Schritt, Transition, Wirkverbindung sowie *kontinuierlich* und *gespeichert wirkenden Aktionen*, die in Tabelle 6-1 zusammengefasst und ihren SIPN-Entsprechungen gegenübergestellt sind. Ein Basic-Grafcet enthält keine *zeitabhängigen Bedingungen*, *Aktionsbedingungen*, *einschließenden Schritte* oder *zwangsstuernden Befehle*. Die nachfolgenden Erörterungen zur formalen Definition eines Basic-Grafcet als SIPN (*formales Modell*) fassen die Ausführungen in [DAVID & ALLA 2010] zusammen und sind gleichzeitig Ausgangspunkt für die Erweiterung des formalen Modells (\rightarrow 6.2).

Die Struktur eines Basic-Grafcet $Grafcet_{Basic}$ kann durch ein binär markiertes ST-Netz beschrieben werden und besteht folglich aus einer abgeschlossenen Menge von Schritten S der Anzahl $|S|$ und einer abgeschlossenen Menge von Transitionen T der Anzahl $|T|$, so dass gilt:

$$S = \{s_1, \dots, s_{|S|}\} \quad (6-1)$$

$$T = \{t_1, \dots, t_{|T|}\} \quad (6-2)$$

S und T sind disjunkte Knotenmengen:

$$T \cap S = \emptyset \tag{6-3}$$

Schritte und Transitionen sind durch Wirkverbindungen miteinander verbunden, die in der sogenannten Flussrelation W zusammengefasst werden:

$$W \subseteq Pre \cup Post = S \times T \cup T \times S \tag{6-4}$$

Die Wirkverbindungen, die von den Schritten zu den Transitionen verlaufen, werden auch als Präkanten, diejenigen, die von den Transitionen zu den Schritten verlaufen auch als Postkanten bezeichnet. Die zugehörigen abgeschlossenen Mengen sind **Pre (Präkanten)** und **Post (Postkanten)**. Wirkverbindungen verlaufen stets von Schritt zu Transition bzw. von Transition zu Schritt und dürfen niemals Elemente aus der gleichen Knotenmenge (S oder T) verbinden. Dementsprechend sind in der Wirkverbindungsrelation die entsprechenden Prä- und Postkanten aufgeführt.

Tabelle 6-1: Grafische Elemente eines Basic-Grafcet und ihre SIPN-Äquivalente

	GRAFCET	SIPN
Schritt, Transition, Wirkverbindung		
Initialschritt		Wird durch Anfangsmarkierung festgelegt
Parallelverzweigung		
Alternativverzweigung		
kontinuierlich wirkende Aktion		
gespeichert wirkende Aktion		

Die Menge der Transitionen, von denen Wirkverbindungen zu einem Schritt s_i bestehen, wird als Vorbereich des Schrittes $\bullet s_i$ bezeichnet. Die Menge der Transitionen, zu denen Wirkverbindungen von einem Schritt s_i verlaufen, wird als Nachbereich des Schrittes $s_i \bullet$ bezeichnet. Entsprechendes gilt für Vorbereich $\bullet t_j$ und Nachbereich $t_j \bullet$ von Transitionen:

$$\bullet s_i = \{t \in \mathbf{T} \mid (t, s_i) \in \mathbf{W}\} \quad s_i \bullet = \{t \in \mathbf{T} \mid (s_i, t) \in \mathbf{W}\} \quad (6-5)$$

$$\bullet t_j = \{s \in \mathbf{S} \mid (s, t_j) \in \mathbf{W}\} \quad t_j \bullet = \{s \in \mathbf{S} \mid (t_j, s) \in \mathbf{W}\} \quad (6-6)$$

Die momentane Situation $\mathbf{sit}(m)$ eines Basic-Grafcet zum Zeitpunkt m ist durch die Menge der zum Zeitpunkt m aktiven Schritte charakterisiert und vergleichbar mit der Markierung eines Petrinetzes. Der Zustand eines einzelnen Schritts s_j aus \mathbf{S} wird durch seine Schrittvariable $X_i(m)$ in $\mathbf{sit}(m)$ repräsentiert, so dass gilt:

$$\mathbf{sit}(m) = \{X_1(m), \dots, X_{/s/}(m)\} \quad (6-7)$$

mit $X_i(m)=0$ für inaktive Schritte s_i , $X_i(m)=1$ für aktive Schritte s_i , $i = 1../\mathbf{S}/$

Die zum Zeitpunkt der Initialisierung aktive Situation (Initialsituation) ist durch die Menge der Initialschritte beschrieben:

$$\mathbf{sit}_0 = \mathbf{sit}(0) = \{X_1(0), \dots, X_{/s/}(0)\} \quad (6-8)$$

In Anlehnung an synchronisierte Petrinetze in [DAVID & ALLA 2010, S.64], zu denen auch SIPN gezählt werden, kann der Wirkungsteil eines Basic-Grafcet durch Eingaben und Ausgaben sowie durch Funktionen beschrieben werden, welche die Verknüpfung der Ein- und Ausgaben mit den Schritten und Transitionen definieren. Die Eingabemenge \mathbf{I} umfasst alle Booleschen und numerischen Eingaben i , die als Bedienereingaben oder Prozessrückmeldungen in einem Grafcet spezifiziert werden. In der Ausgabemenge \mathbf{O} werden alle Booleschen und numerischen Bedienerrückmeldungen und Prozessausgaben o der Steuerungseinheit des Grafcet zusammengefasst. Zählvariablen oder Merker var sind der Variablenmenge \mathbf{Var} zugeordnet und repräsentieren die Rückgaben und Anweisungen der Verarbeitungseinheit. Die konkreten Funktionen zur Verknüpfung der Ein- und Ausgaben aus \mathbf{I} , \mathbf{O} und \mathbf{Var} mit der Struktur eines Grafcet werden gemäß IEC 60848 durch Transitionsbedingungen $r(t)$ und Aktionen $a(s)$ beschrieben.

Transitionsbedingungen sind den Transitionen zugeordnet und werden in der Menge der Transitionsbedingungen aufgeführt:

$$\mathbf{r} = \{r(t_1), \dots, r(t_{/T/})\} \quad (6-9)$$

Eine zu einer Transition t_j zugehörige Transitionsbedingung $r(t_j) \in \mathbf{r}$ setzt sich zusammen aus einem Ereignis E_j und einer Bedingung B_j mit

$$r(t_j) = E_j \wedge B_j, j = 1../\mathbf{T}/ \quad (6-10)$$

wobei für E_j und B_j gilt:

$$E_j = f: \{\uparrow, \downarrow\}^{I, \mathbf{sit}} \rightarrow t \quad (6-11)$$

$$B_j = f: \{0,1\}^{I, \mathbf{Var}, \mathbf{sit}} \rightarrow t \quad (6-12)$$

Ein Ereignis E_j ist eine Funktion $f: \{\uparrow, \downarrow\}^{I, \mathbf{sit}} \rightarrow t$, die jeder Transition $t_j \in \mathbf{T}$ einen Ausdruck der Eingaben aus \mathbf{I} und der Schrittvariablen aus \mathbf{sit} zuordnet, der als Ereignis gemäß der *Algebra of Events* ausgewertet wird. Eine Bedingung ist eine Boolesche

Funktion $f: \{0,1\}^I, \mathbf{Var}, \mathbf{sit} \rightarrow t$, die jeder Transition aus \mathbf{T} eine Bedingung der Eingaben aus \mathbf{I} und \mathbf{Var} sowie der Schrittvariablen aus \mathbf{sit} zuordnet, die als logische Aussage ausgewertet wird. Für jede Transition $t_j \in \mathbf{T}$ existiert eine entsprechende Transitionsbedingung $r(t_j)$, die nicht konstant *false* sein darf, wobei die immer erfüllte Bedingung durch $B_j = \text{true}$ und das immer eintretende Ereignis durch $E_j = \varepsilon$ festgelegt sind. Dies bedeutet, dass die der Transitionsbedingung $r(t)$ zugeordnete logische Variable $[r(t)]$ entweder den Wert $[r(t)] = 1$ ($=\text{true}$) oder den Wert $[r(t)] = 0$ ($=\text{false}$) annehmen oder immer erfüllt sein kann. Es gilt $[r(t_j)] = 1$, wenn die Bedingung B_j erfüllt ist und das Ereignis E_j eintritt.

Aktionen sind über Assoziationen mit Schritten verbunden und in der Menge der Aktionen

$$\mathbf{a} = \mathbf{a}_{\text{kont}} \cup \mathbf{a}_{\text{gesp}} \quad (6-13)$$

enthalten, die sich aus der Menge der *kontinuierlich wirkenden Aktionen* \mathbf{a}_{kont} und der Menge der *gespeichert wirkenden Aktionen* \mathbf{a}_{gesp} zusammensetzt:

$$\mathbf{a}_{\text{kont}} = \{a_{\text{kont},1}, \dots, a_{\text{kont},p}\}, p: \text{Anzahl der kontinuierlich wirkenden Aktionen} \quad (6-14)$$

$$\mathbf{a}_{\text{gesp}} = \{a_{\text{gesp},1}, \dots, a_{\text{gesp},q}\}, q: \text{Anzahl der gespeichert wirkenden Aktionen} \quad (6-15)$$

$$\mathbf{a}_{\text{kont}} \cap \mathbf{a}_{\text{gesp}} = \emptyset \quad (6-16)$$

Die maximale Anzahl der *kontinuierlich* und *gespeichert wirkenden Aktionen* ist gemäß den Definitionen der IEC 60848 nicht begrenzt. Die Zuordnung der Ausgaben aus \mathbf{O} zur jeweiligen Aktionsart muss stets eindeutig sein. Eine *kontinuierlich wirkende Aktion* $a_{\text{kont},p}(s_i) \in \mathbf{a}_{\text{kont}}$ ist eine Boolesche Funktion $f: \{0,1\}^{\mathbf{O}} \rightarrow s$, die einem Schritt s_i Boolesche Bedienerückmeldungen und Prozessausgaben aus \mathbf{O} zuordnet, so dass gilt:

$$a_{\text{kont},p}(s_i): \begin{cases} 1, & \text{wenn } (X_i = 1) \\ 0, & \text{sonst} \end{cases} \quad (6-17)$$

Eine *gespeichert wirkende Aktion* $a_{\text{gesp},q}(s_i) \in \mathbf{a}_{\text{gesp}}$ ist entweder eine Boolesche Funktion $f: \{0,1\}^{\mathbf{O}, \mathbf{Var}} \rightarrow \{\uparrow X_i, \downarrow X_i\}$, oder eine Operation auf dem numerischen Wertebereich \mathbf{V} , $f: \{\mathbf{V}\}^{\mathbf{O}, \mathbf{Var}} \rightarrow \{\uparrow X_i, \downarrow X_i\}$, die einmalig und gespeichert ausgeführt wird, wenn das mit dem Schritt s_i verknüpfte Ereignis $\uparrow X_i$ bzw. $\downarrow X_i$ eintritt:

$$a_{\text{gesp},q}(s_i): \begin{cases} f: \{0,1\}^{\mathbf{O}, \mathbf{Var}}, & \text{wenn } \{\uparrow X_i, \downarrow X_i\} \\ f: \{\mathbf{V}\}^{\mathbf{O}, \mathbf{Var}}, & \text{wenn } \{\uparrow X_i, \downarrow X_i\} \end{cases} \quad (6-18)$$

Entgegen der *kontinuierlich wirkenden Aktionen* bleibt der Wert, welcher der Variablen der Verarbeitungseinheit in \mathbf{Var} oder der Booleschen oder numerischen Bedienerückmeldung bzw. Prozessausgabe in \mathbf{O} durch $a_{\text{gesp},q}(s_i)$ zugewiesen wird, bei *gespeichert wirkenden Aktionen* auch dann erhalten, also gespeichert, wenn s_i deaktiviert ist.

Das charakteristische dynamische Verhalten eines Basic-Grafcet umfasst einerseits den Übergang von der momentanen Situation $\mathbf{sit}(m)$ zur nächstfolgenden Situation $\mathbf{sit}(m+1)$ und andererseits die Ausführung der *kontinuierlich* und *gespeichert wirkenden Aktionen* entsprechend der Spezifikation. Für markierte Petrinetze, wie beispielsweise in [LITZ 2005, S.228-232] aufgezeigt, kann die Nachfolgemarkierung $\mathbf{m}(k+1)$ mit Hilfe einer Inzidenzmatrix \mathbf{C} und mit Kenntnis der aktuellen Markierung $\mathbf{m}(k)$ sowie der zum Zeitpunkt k feuern den Transitionen berechnet werden. Unter Berücksichtigung der spezifischen

Eigenschaften von GRAFCET ist eine entsprechende Definition zur Berechnung der Nachfolgesituation $\mathbf{sit}(m+1)$ eines Basic-Grafcet durch die Inzidenzmatrix $\mathbf{C}_{\text{basic}}$ möglich. Die Berechnung der Nachfolgesituation ist wie folgt möglich:

$$\mathbf{sit}(m+1) = \mathbf{sit}(m) + \mathbf{C}_{\text{basic}} \cdot \boldsymbol{\delta}(m) \quad (6-19)$$

$$\text{mit } \mathbf{sit}^T(m) = [X_1(m), \dots, X_{/S/}(m)]$$

$/S/ \times 1$ -Vektor der aktuellen GRAFCET-Situation zum Zeitpunkt m ,

$$\mathbf{sit}^T(m+1) = [X_1(m+1), \dots, X_{/S/}(m+1)]$$

$/S/ \times 1$ -Vektor der nachfolgenden GRAFCET-Situation zum Zeitpunkt $m+1$,

Für die Elemente c_{ij} der $/S/ \times /T/$ -Matrix $\mathbf{C}_{\text{basic}}$ gilt:

$$c_{ij} = \begin{cases} -1, & \text{wenn } (s_i, t_j) \in \mathbf{Pre} \\ 1, & \text{wenn } (t_j, s_i) \in \mathbf{Post} \\ 0, & \text{sonst} \end{cases} \quad (6-20)$$

Die Präkanten eines Basic-Grafcet werden mit dem Gewicht -1, die Postkanten mit dem Gewicht 1 in der Inzidenzmatrix dokumentiert. Alle anderen Elemente von $\mathbf{C}_{\text{basic}}$ sind gleich Null. Für $\boldsymbol{\delta}^T(m)$ gilt:

$$\boldsymbol{\delta}^T(m) = [\delta_1(m), \dots, \delta_j(m)] \quad (6-21)$$

Der $/T/ \times 1$ -Vektor $\boldsymbol{\delta}^T(m)$ enthält für jede Transition t_j aus T eine logische Variable $\delta_j(m)$, deren Wert 1 ist, wenn die Transition zum Zeitpunkt m auslöst. t_j löst genau dann aus, wenn alle Schritte im Vorbereich $\bullet t_j$ aktiv sind und die Transitionsbedingung $[r(t_j)]$ erfüllt ist:

$$\delta_j(m) \begin{cases} 1 \Leftrightarrow ((\forall s_i \in \bullet t_j : X_i = 1) \wedge [r(t_j)] = 1) \\ 0 \Leftrightarrow \text{sonst} \end{cases} \quad (6-22)$$

Die Situation $\mathbf{sit}(m+1)$ eines Grafcet ist erreicht, wenn sie stabil ist. Die Nachfolgesituation ist genau dann stabil, wenn keine der Transitionen im Nachbereich der aktiven Schritte der Nachfolgesituation mehr aufgrund des Ereignisses zum Zeitpunkt m auslösen kann:

$$\mathbf{sit}(m+1) \begin{cases} \text{stabil} \Leftrightarrow (\forall t_j \in s_i \bullet \mid X_i(m+1) = 1) : \delta_j(m) = 0 \\ \text{transient} \Leftrightarrow \text{sonst} \end{cases} \quad (6-23)$$

Sollte es dennoch Transitionen geben, die aufgrund des Ereignisses zum Zeitpunkt m und nach Berechnung mit (6-19) auslösen können, so handelt es sich um eine instabile GRAFCET-Situation bzw. transienten Ablauf (\rightarrow 4.2.3). Die Berechnung muss dann nochmals durchgeführt werden, bis eine stabile Situation für den Zeitpunkt $m+1$ erreicht ist. Da Zustandsübergänge von einer stabilen Situation zur nächsten in GRAFCET unendlich schnell erfolgen, können zwischenzeitlich keine weiteren Ereignisse eintreten. Sollte die Berechnung gemäß (6-19) für einzelne Elemente $\text{sit}_i(m+1)$ von $\mathbf{sit}(m+1)$ negative Werte oder Werte $\text{sit}_i(m+1) > 1$ ergeben, so gilt:

$$\mathbf{sit}(m+1) = \begin{bmatrix} \text{sit}_1(m+1) \\ \vdots \\ \text{sit}_i(m+1) \end{bmatrix} : \begin{cases} \text{sit}_i(m+1) < 0 \Rightarrow \text{sit}_i(m+1) := 0 \\ \text{sit}_i(m+1) > 1 \Rightarrow \text{sit}_i(m+1) := 1 \end{cases} \quad (6-24)$$

Dies resultiert aus den Booleschen Markierungen der GRAFCET-Schritte. Die Ausführung der GRAFCET-Aktionen für den Zeitpunkt $m + 1$ wird durch eine Ausgabefunktion Ω unter Berücksichtigung der Ausführungsbedingungen der jeweiligen Aktionsart aus (6-17) und (6-18) berechnet:

$$\Omega(m + 1) = \prod_{\substack{/S/ \\ \bar{i}=1, \\ \uparrow X_i \vee \downarrow X_i}} \prod_{j=1}^q a_{\text{gesp},j}(S_i) \wedge \prod_{\substack{/S/ \\ \bar{i}=1, \\ X_i(m+1)=1 \wedge \\ \text{stabil}}} \prod_{j=1}^p a_{\text{kont},j}(S_i) \quad (6-25)$$

Kontinuierlich wirkende Aktionen werden nur dann ausgeführt, wenn es sich bei der Nachfolgesituation zum Zeitpunkt $m + 1$ um eine stabile Situation handelt. *Gespeichert wirkende Aktionen* werden hingegen auch für transiente Zwischensituationen ausgeführt.

6.1.1.1 Beispiel eines Basic-Grafcet

Die formalen Definitionen bezüglich des dynamischen Verhaltens sollen anhand des in Abbildung 6-2 ersichtlichen Grafcets noch einmal nachvollzogen werden. Die aktuelle Situation des Grafcet sei die Initialsituation:

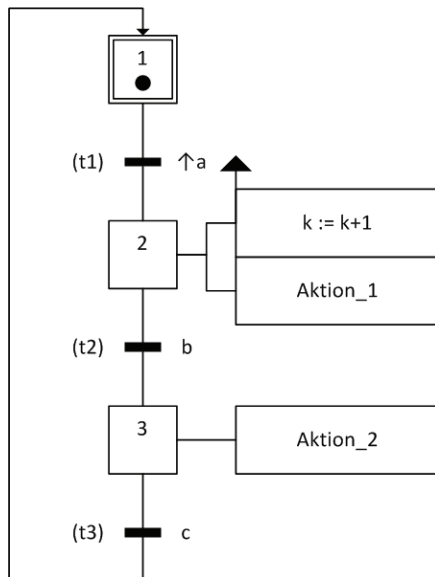


Abbildung 6-2: Beispiel eines Basic-Grafcet

$$\mathbf{sit}(0) = \mathbf{sit}_0 = \{1, 0, 0\},$$

$$\mathbf{sit}^T(0) = \mathbf{sit}^T_0 = [1, 0, 0] \quad (6-26)$$

Für die Transitionsbedingungen soll zunächst gelten:

$$[r(t_1)] = [r(t_2)] = [r(t_3)] = 0,$$

$$\boldsymbol{\delta}^T(m) = [0, 0, 0] \quad (6-27)$$

Der Wert der Zählvariablen k zum Zeitpunkt der Initialisierung beträgt Null. Die Inzidenzmatrix des Grafcet lautet:

$$\mathbf{C}_{\text{basic}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \quad (6-28)$$

Tritt nun das Ereignis $\uparrow a$ ein, so löst die freigegebene Transition t_1 aufgrund $[r(t_1)] = 1$ aus, deaktiviert Schritt s_1 und aktiviert Schritt s_2 . Diese Zustandsänderung des Grafcet kann im formalen Modell auch mit Hilfe von (6-19) berechnet werden. Aufgrund des Ereignisses $\uparrow a$ ergibt sich:

$$\mathbf{sit}(1) = \mathbf{sit}_0 + \mathbf{C}_{\text{basic}} \cdot \boldsymbol{\delta}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad (6-29)$$

Die *gespeichert wirkende Aktion* $a_{\text{gesp},1}(s_2) = [k := k + 1]$ wird einmalig ausgeführt, wenn Schritt s_2 aktiviert wird, die *kontinuierlich wirkende Aktion* $a_{\text{kont},1}(s_2) = \text{Aktion}_1$ wird ausgeführt, solange Schritt s_2 aktiv ist.

Somit ergibt sich für die Ausgabe:

$$\mathbf{\Omega}(1) = a_{\text{gesp},1}(s_2) \wedge a_{\text{kont},1}(s_2) \quad (6-30)$$

Zur Demonstration transienter Abläufe soll nun folgende Situation angenommen werden:

$$\mathbf{sit}^T(0) = \mathbf{sit}^{T_0} = [1, 0, 0] \quad (6-31)$$

$$[r(t_1)] = [r(t_3)] = 0, [r(t_2)] = 1$$

Tritt nun das Ereignis $\uparrow a$ ein, so ergibt sich für die nachfolgende Situation zunächst dasselbe Ergebnis wie im Fall von (6-29) und (6-30):

$$\mathbf{sit}^T(1)' = [0, 1, 0], \mathbf{\Omega}(1)' = a_{\text{gesp},1}(s_2) \wedge a_{\text{kont},1}(s_2) \quad (6-32)$$

Diese berechnete Situation ist allerdings instabil, da die Transitionsbedingung $r(t_2)$ gemäß den Vorgaben zum Zeitpunkt $m = 0$ erfüllt ist. Durch die Aktivierung von Schritt s_2 löst nun gleichzeitig t_2 aus, so dass eine weitere Berechnung gemäß (6-19) erforderlich ist:

$$\mathbf{sit}(1) = \mathbf{sit}(1)' + \mathbf{C}_{\text{basic}} \cdot \mathbf{\delta}(1)' = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (6-33)$$

Die nächstfolgende Situation des Grafcet ist dann $\mathbf{sit}^T(1) = [0, 0, 1]$.

Hierbei handelt es sich nun um eine stabile Situation gemäß (6-23). Für die Ausgabe ergibt sich in dieser Situation:

$$\mathbf{\Omega}(1) = a_{\text{kont},2}(s_3) \quad (6-34)$$

Hierbei ist zu beachten, dass die *gespeichert wirkende Aktion* $a_{\text{gesp},1}(s_2)$ aufgrund ihrer Verknüpfung mit dem Ereignis $\uparrow X_2$ trotzdem wirksam wird. Die *kontinuierlich wirkende Aktion* $a_{\text{kont},1}(s_2)$ wird hingegen nicht ausgeführt.

Diese zuvor erläuterten Zusammenhänge bezüglich des Basic-Grafcet bilden das Grundgerüst für die formale Definition von GRAFCET als SIPN. Alle in den nachfolgenden Kapiteln beschriebenen weiteren GRAFCET-Elemente bieten die Möglichkeit, einen Grafcet kompakter zu gestalten, lassen sich aber auch durch einen äquivalenten Basic-Grafcet beschreiben. Der Basic-Grafcet stellt somit eine **Normalform** dar, auf die alle erweiterten GRAFCET-Konstrukte formal zurückgeführt werden können. Die normalisierte Form wird bisher aber nur für Makroschritte in [DAVID & ALLA 2010] erläutert (\rightarrow 6.1.2). Für *einschließende Schritte*, *zwangsteuernde Befehle* und *zeitabhängige Bedingungen* ist eine entsprechende formale Definition bisher nicht bekannt und wird im Rahmen dieser Arbeit zur Erweiterung und Vervollständigung des formalen Modells erstellt.

6.1.3 Berücksichtigung von Makroschritten

In einem Grafcet dienen Makroschritte grundsätzlich als Platzhalter für eine detaillierte Sicht auf die Steuerungsabläufe in einem konkreten Schritt (\rightarrow 4.3.1). In der groben Darstellung des Grafcet ist ein Makroschritt enthalten, wie beispielsweise Schritt s_{M1} links in Abbildung 6-3, der wie ein Schritt in die GRAFCET-Struktur eingebunden wird. Der Makroschritt verweist auf eine Feinstruktur, die an Stelle des Makroschrittes in den Grafcet eingebunden werden kann, in Abhängigkeit des gewünschten Detaillierungsgrads der Spezifikation.

Für die Integration des Makroschritt-Konzepts in das formale SIPN-Modell für GRAFCET ergeben sich zwei unterschiedliche Möglichkeiten. Zum einen kann der Makroschritt in einem Grafcet als reiner Platzhalter verwendet werden, ohne dass gleichzeitig eine Feinstruktur spezifiziert wird. Die Feinstruktur würde demzufolge zu einem späteren Zeitpunkt, wie beispielsweise im Zuge einer Detailplanung, erstellt. Im formalen Modell wird der Makroschritt dann wie ein Schritt berücksichtigt, wobei die abgeschlossene Menge der Makroschritte S_{Makro} eine echte Teilmenge von S ist:

$$S_{\text{Makro}} \subseteq S \tag{6-35}$$

mit $S_{\text{Makro}} = \{s_{\text{Makro},1}, \dots, s_{\text{Makro},k}\}$, k: Anzahl der Makroschritte

Weitere Auswirkungen auf das formale Modell ergeben sich nicht, so dass die bisherigen Definitionen (\rightarrow 6.1.1) ausreichend sind.

Zum anderen kann der Makroschritt in einem Grafcet aus Gründen der Übersichtlichkeit verwendet werden. In diesem Fall existiert eine Feinstruktur, wie im linken Teil von Abbildung 6-3 für s_{M1} aufgezeigt. Im formalen Modell wird an Stelle des Makroschrittes die zugehörige Feinstruktur berücksichtigt, die sich direkt in die Struktur des Grafcet integrieren

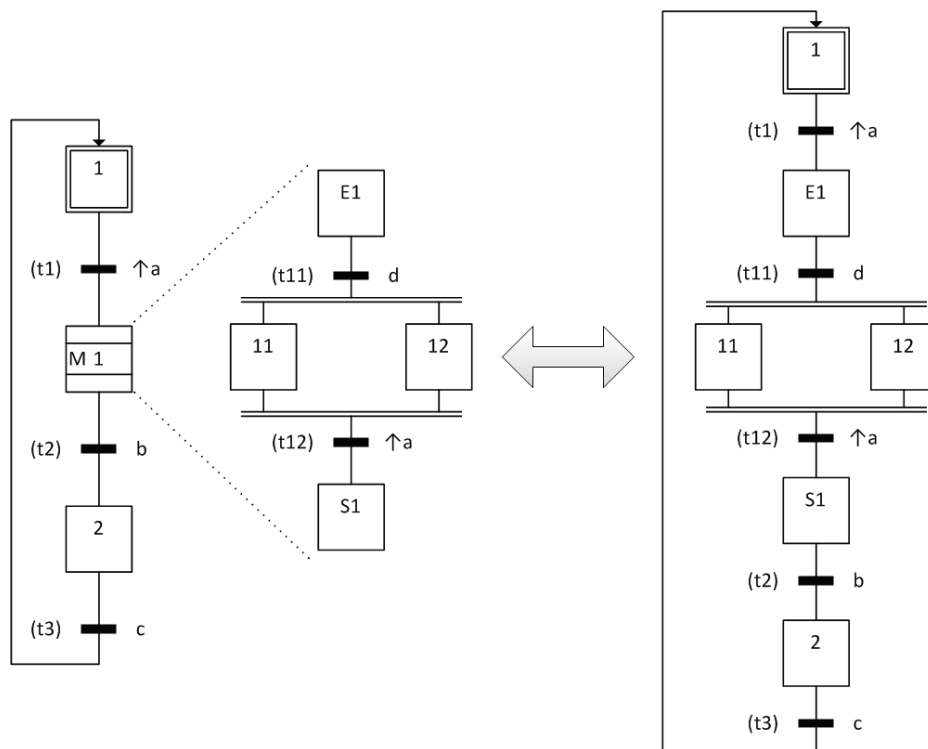


Abbildung 6-3: Struktur eines Grafcet mit Makroschritt (links) und Normalform (rechts)

lässt. Die resultierende äquivalente Struktur ist beispielhaft im rechten Teil von Abbildung 6–3 dargestellt. Somit ergibt sich im angegebenen Beispiel für die Inzidenzmatrix:

$$\mathbf{C} = \begin{bmatrix} C_{1,1} & C_{1,11} & C_{1,12} & C_{1,2} & C_{1,3} \\ C_{E1,1} & C_{E1,11} & C_{E1,12} & C_{E1,2} & C_{E1,3} \\ C_{11,1} & C_{11,11} & C_{11,12} & C_{11,2} & C_{11,3} \\ C_{12,1} & C_{12,11} & C_{12,12} & C_{12,2} & C_{12,3} \\ C_{S1,1} & C_{S1,11} & C_{S1,12} & C_{S1,2} & C_{S1,3} \\ C_{2,1} & C_{2,11} & C_{2,12} & C_{2,2} & C_{2,3} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \quad (6-36)$$

Weitere Anpassungen sind nicht notwendig, so dass beide Möglichkeiten zur Spezifikation mit Makroschritten einfach in das formale Modell integriert werden können.

6.2 Erweiterung des formalen Modells

6.2.1 Kontinuierlich und gespeichert wirkende Aktionen

Kontinuierlich und *gespeichert wirkende Aktionen* wurden im formalen Modell bisher nur in ihrer grundlegenden Form berücksichtigt. Darüber hinaus definiert die IEC 60848 Zuweisungsbedingungen, die als zusätzliche Voraussetzung für die Ausführung *kontinuierlich wirkender Aktionen* spezifiziert werden können. Die Ausführung einer *kontinuierlich wirkenden Aktion* mit Zuweisungsbedingung ist sowohl vom momentanen Aktivierungszustand des Schrittes als auch vom aktuellen Zustand der Zuweisungsbedingung abhängig. Das Beispiel in Abbildung 6-4 zeigt hierzu eine dem Schritt s_4 assoziierte *kontinuierlich wirkende Aktion* $a_{\text{kont},1}(s_4) = \text{"Ventil_1, wenn } a\text{"}$ mit der Zuweisungsbedingung a , die durch eine explizite grafische Kennzeichnung hervorgehoben wird. $a_{\text{kont},1}(s_4)$ wird genau dann ausgeführt, wenn s_4 aktiv ($X_4 = 1$) und a erfüllt ist. Sollte im weiteren Verlauf entweder a nicht mehr erfüllt sein oder s_4 deaktiviert werden ($X_4 = 0$), so wird ebenfalls $a_{\text{kont},1}(s_4)$ deaktiviert. Eine Zuweisungsbedingung $Cond_{\text{bool}}$ ist somit eine Boolesche Funktion $f: \{0,1\}^I, \text{Var. } X \rightarrow a_{\text{kont},p}(s_i)$, die jeder *kontinuierlich wirkenden Aktion* $a_{\text{kont},p}(s_i) \in \mathbf{a}_{\text{kont}}$ eine Bedingung der Eingaben aus I und Var sowie der Schrittvariablen aus X zuordnet, die als logische Aussage ausgewertet wird. Die immer erfüllte Zuweisungsbedingung ist $Cond_{\text{bool}}(a_{\text{kont},p}(s_i)) = \text{true}$. Die der Zuweisungsbedingung zugeordnete logische Variable besitzt, abhängig vom aktuellen Zustand des Grafcet, entweder den Wert $[Cond_{\text{bool}}(a_{\text{kont},p}(s_i))] = 1$ bzw. $[Cond_{\text{bool}}(a_{\text{kont},p}(s_i))] = 0$ oder ist immer erfüllt.

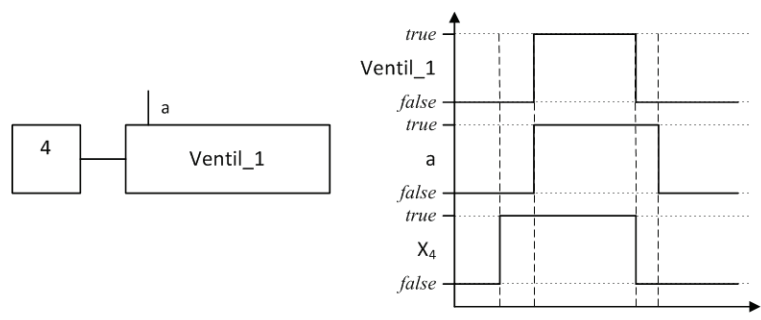


Abbildung 6-4: Kontinuierlich wirkende Aktion mit Zuweisungsbedingung

Folglich muss die Bedingung für das Ausführen einer *kontinuierlich wirkenden Aktion* in (6-17) entsprechend angepasst werden, so dass gilt:

$$a_{\text{kont},p}(S_i): \begin{cases} 1, & \text{wenn } (X_i = 1) \wedge (Cond_{\text{bool},p} = 1) \\ 0, & \text{sonst} \end{cases} \quad (6-37)$$

Auch *gespeichert wirkende Aktionen* können gemäß den Definitionen der IEC 60848 nicht nur mit der Aktivierung oder Deaktivierung des assoziierten Schrittes, sondern alternativ auch mit beliebigen Ereignissen verknüpft werden. Diese sogenannte *Aktion bei Ereignis* wird grafisch hervorgehoben (\rightarrow Abbildung 6-5) und genau dann gespeichert ausgeführt, wenn der assoziierte Schritt s_4 aktiv ist ($X_4 = 1$) und das Ereignis $\downarrow a$ eintritt. Sollte $\downarrow a$ nicht eintreten, solange Schritt ($X_4 = 1$) gilt, so wird die *gespeichert wirkende Aktion* $a_{\text{gesp},1}(s_4) = k := 5$ nicht ausgeführt.

Eine *Aktion bei Ereignis* ist demzufolge eine *gespeichert wirkende Aktion* $a_{\text{gesp},q}(S_i) \in \mathbf{a}_{\text{gesp}}$ in Form einer Booleschen Funktion, $f: \{0,1\}^{\mathcal{O}, \text{Var}} \rightarrow \{\uparrow, \downarrow\}^{\mathcal{L}, \mathcal{X}}$, oder eine Operation auf dem numerischen Wertebereich \mathbf{V} , $f: \{\mathbf{V}\}^{\mathcal{O}, \text{Var}} \rightarrow \{\uparrow, \downarrow\}^{\mathcal{L}, \mathcal{X}}$, die einmalig und gespeichert ausgeführt wird, wenn das aus den Eingaben \mathbf{I} und den Schrittvariablen aus \mathbf{X} spezifizierte Ereignis $\{\uparrow, \downarrow\}^{\mathcal{L}, \mathcal{X}}$ eintritt:

$$a_{\text{gesp},q}(S_i): \begin{cases} f: \{0,1\}^{\mathcal{O}, \text{Var}}, & \text{wenn } \{\uparrow, \downarrow\}^{\mathcal{L}, \mathcal{X}} \\ f: \{\mathbf{V}\}^{\mathcal{O}, \text{Var}}, & \text{wenn } \{\uparrow, \downarrow\}^{\mathcal{L}, \mathcal{X}} \end{cases} \quad (6-38)$$

Die *Aktion bei Ereignis* ist der verallgemeinerte Fall einer *gespeichert wirkenden Aktion* und stellt eine Erweiterung der Definitionen aus (6-18) dar.

Insgesamt wirkt sich die Erweiterung des Wirkungsteils in Form von Zuweisungsbedingungen und Aktionen bei Ereignis auch auf das dynamische Verhalten eines Grafcet aus, so dass die Ausgabefunktion in (6-25) gemäß den Definitionen in (6-37) und (6-38) erweitert wird zu:

$$\Omega(m+1) = \prod_{\substack{i=1, \\ \{\uparrow, \downarrow\}^{\mathcal{L}, \mathcal{X}}}}^{/S/} \prod_{j=1}^q a_{\text{gesp},j}(S_i) \wedge \prod_{\substack{i=1, \\ X_i(m+1)=1 \wedge \\ \text{stabil}}}^{/S/} \prod_{\substack{j=1, \\ Cond_{\text{bool}} \\ (a_{\text{kont},p}(S_i))=1}}^p a_{\text{kont},j}(S_i) \quad (6-39)$$

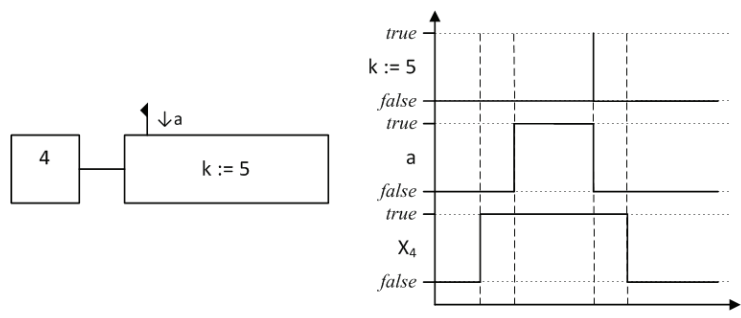


Abbildung 6-5: Aktion bei Ereignis

6.2.2 Einschließende Schritte

Durch das GRAFCET-Konstrukt der Einschließung, bestehend aus einem *einschließenden Schritt* und dem Teil-Grafcet der eingeschlossenen Schritte, können Abhängigkeiten zwischen Teil-Grafcets in kompakter Form spezifiziert werden (\rightarrow 4.3.2). Charakteristisch bezüglich des dynamischen Verhaltens von Einschließungen ist, dass der Zustand des Teil-Grafcets der eingeschlossenen Schritte unmittelbar vom aktiven oder inaktiven Zustand des *einschließenden Schrittes* abhängig ist. Diese Abhängigkeit resultiert aus sogenannten *impliziten Wirkverbindungen*, die definitionsgemäß durch die Spezifikation mit *einschließenden Schritten* wirksam werden. Durch die Transformation des nunmehr hierarchisch strukturierten Grafcets in die Normalform werden diese *impliziten Wirkverbindungen* sozusagen sichtbar und sind nicht mehr nur in der Symbolik des *einschließenden Schritts* verborgen.

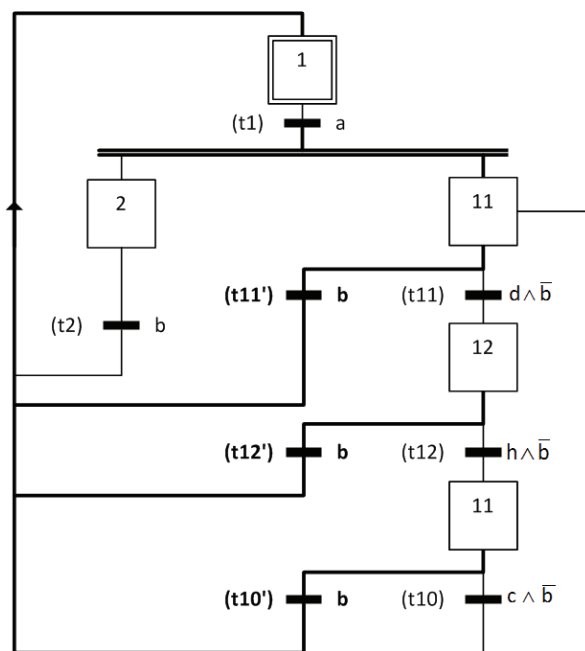


Abbildung 6-6: Normalform der Einschließung aus Abbildung 4-5

Abbildung 6-6 zeigt den normalisierten Grafcet G_N für den Grafcet aus Abbildung 4-5, anhand dessen das Prinzip der Einschließung erläutert wurde. Die in G_N sichtbaren zusätzlichen Wirkverbindungen und Transitionen sind grafisch hervorgehoben und verlaufen von jedem Schritt s^E des Teil-Grafcets der eingeschlossenen Schritte G^E zu den Transitionen $t_{11'}$, $t_{12'}$, $t_{13'}$. Diese zusätzlichen Transitionen resultieren aus den *impliziten Wirkverbindungen* der Einschließung und sind in ihren Transitionsbedingungen identisch zur Transition im Nachbereich $s_E \bullet$ des *einschließenden Schrittes*. Durch sie wird verdeutlicht, dass im Falle des Aus-

lösens von t_2 nicht nur $s_E = s_2$, sondern auch alle eingeschlossenen Schritte $\{s_{10}, s_{11}, s_{12}\}$ deaktiviert werden. Sollte $s_E \bullet$ mehrere Transitionen umfassen, so existiert für jeden Schritt s^E je eine *implizite Wirkverbindung* zu den Transitionen in $s_E \bullet$. In diesem Falle ist besonders darauf zu achten, dass der Grafcet weiterhin die Anforderungen an einen *sound Grafcet* (\rightarrow 6.1.1) erfüllt. Auch im Beispiel aus Abbildung 6-6 muss die Anforderung sich gegenseitig ausschließender Transitionsbedingungen berücksichtigt werden. Daher muss für jede Transition des Teil-Grafcets der eingeschlossenen Schritte t^E die entsprechende Transitionsbedingung $r(t^E)$ durch die negierte Transitionsbedingung der Transitionen aus $s_E \bullet$ im Sinne einer Konjunktion erweitert werden. Die Transitionsbedingungen mehrerer Transitionen in $s_E \bullet$ werden durch Disjunktion miteinander verknüpft.

Für das Beispiel in Abbildung 6-6 muss für die Transitionsbedingungen $r(t_{10})$, $r(t_{11})$ und $r(t_{12})$ folglich gelten:

$$r(t_{10}) = c \wedge \bar{b} \quad (6-40)$$

$$r(t_{11}) = d \wedge \bar{b} \quad (6-41)$$

$$r(t_{12}) = h \wedge \bar{b} \quad (6-42)$$

Weitere *implizite Wirkverbindungen* resultieren aus der Aktivierungsverbindung der Einschließung, die festlegt, welche Schritte gleichzeitig mit s_E aktiviert werden. Im Beispiel aus Abbildung 6-6 führt das Auslösen von t_1 , aufgrund der bestehenden Aktivierungsverbindung, zu einer simultanen Aktivierung von s_2 und s_{11} . Aus der Perspektive des formalen SIPN-Modells für GRAFCET ergeben sich durch *einschließende Schritte* die nachfolgend erläuterten strukturellen Erweiterungen.

Die Menge der eingeschlossenen Schritte \mathcal{S}_E ist eine echte Teilmenge von \mathcal{S} :

$$\mathcal{S}_E \subseteq \mathcal{S} \quad (6-43)$$

mit $\mathcal{S}_E = \{s_{E,1}, \dots, s_{E,m}\}$, m: Anzahl der *einschließenden Schritte*

Mit jedem *einschließenden Schritt* s_E ist unmittelbar ein Teil-Grafcet der eingeschlossenen Schritte G^E verknüpft, dessen zugehörige Schrittvariablen durch die abgeschlossene Menge X^E beschrieben sind. Die Menge der eingeschlossenen Schritte ist \mathcal{S}^E , die Menge der mit einer Aktivierungsverbindung versehenen eingeschlossenen Schritte ist $\mathcal{S}^{E*} \subseteq \mathcal{S}^E$.

Die s_E zugeordnete Einschließung $\varphi(s_E) \in \boldsymbol{\varphi}$ ist eine Boolesche Funktion, $f: \{0,1\}^{X^E} \rightarrow \{0,1\}$, die allen Schritten aus \mathcal{S}^{E*} den aktiven Schrittzustand zuweist (Aktivierungsverbindung), wenn das Ereignis $\uparrow X_E$ eintritt und allen Schritten aus \mathcal{S}^E den inaktiven Zustand zuweist, wenn das Ereignis $\downarrow X_E$ eintritt:

$$\varphi(s_{E,m}): \begin{cases} f: \{0,1\}^{X^{E*}} \rightarrow \{1\}, \text{ wenn } \uparrow X_{E,m} \\ f: \{0,1\}^{X^E} \rightarrow \{0\}, \text{ wenn } \downarrow X_{E,m} \end{cases} \quad (6-44)$$

Solange der *einschließende Schritt* aktiv ist ($X_E = 1$), besitzen alle eingeschlossenen Schritte $s^E \in \mathcal{S}^E$ die Eigenschaft eines Schrittes.

Die Auswirkungen *einschließender Schritte* auf das dynamische Verhalten zeigen sich hinsichtlich des formalen Modells anhand der Inzidenzmatrix von G_N . Dies soll nachfolgend anhand des Beispiels von Abbildung 4-5 und Abbildung 6-6 und veranschaulicht werden. Es werden dazu zunächst die Inzidenzmatrizen \mathbf{C}_1 und \mathbf{C}_{10} der beiden Teil-Grafcets G_1 und G_{10} aus Abbildung 4-5 unabhängig voneinander gebildet. \mathbf{C}_1 und \mathbf{C}_{10} sind wie folgt aufgebaut:

$$\mathbf{C}_1 = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad (6-45)$$

$$\mathbf{C}_{10} = \begin{bmatrix} c_{10,10} & c_{10,11} & c_{10,12} \\ c_{11,10} & c_{11,11} & c_{11,12} \\ c_{12,10} & c_{12,11} & c_{12,12} \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \quad (6-46)$$

Durch den *einschließenden Schritt* s_2 werden G_1 und G_{10} nun so voneinander abhängig, dass \mathbf{C}_1 und \mathbf{C}_{10} in einer globalen Matrix des normalisierten Grafcet \mathbf{C}_N wie folgt zusammengeführt werden können:

$$\mathbf{C}_N = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,10} & c_{1,11} & c_{1,12} & c_{1,10'} & c_{1,11'} & c_{1,12'} \\ c_{2,1} & c_{2,2} & c_{2,10} & c_{2,11} & c_{2,12} & c_{2,10'} & c_{2,11'} & c_{2,12'} \\ c_{10,1} & c_{10,2} & c_{10,10} & c_{10,11} & c_{10,12} & c_{10,10'} & c_{10,11'} & c_{10,12'} \\ c_{11,1} & c_{11,2} & c_{11,10} & c_{11,11} & c_{11,12} & c_{11,10'} & c_{11,11'} & c_{11,12'} \\ c_{12,1} & c_{12,2} & c_{12,10} & c_{12,11} & c_{12,12} & c_{12,10'} & c_{12,11'} & c_{12,12'} \end{bmatrix} \quad (6-47)$$

Die durch die Normalisierung resultierenden zusätzlichen Wirkverbindungen und Transitionen (\rightarrow Abbildung 6-6) lassen sich in Form einer Inzidenzmatrix der Einschließungen \mathbf{C}_φ zusammenfassen. Für die Werte der Elemente $c^{\varphi_{ij}}$ gelten die folgenden Bedingungen für die *impliziten Wirkverbindungen*:

$$c^{\varphi_{ij}} = \begin{cases} -1 \Leftrightarrow \forall (s^{E_i}, t_j) \in \mathbf{Pre} | ((t_j \in s_{E,m} \bullet) \wedge (s^{E_i} \in \mathbf{S}^E)) \\ 1 \Leftrightarrow \exists (t_j, s^{E_i}) \in \mathbf{Post} | ((t_j \in \bullet s_{E,m}) \wedge (s^{E_i} \in \mathbf{S}^{E*})) \\ 0, \text{ sonst} \end{cases} \quad (6-48)$$

Es gilt $c^{\varphi_{ij}} = -1$ für alle Präkanten, die als *implizite Wirkverbindungen* von den eingeschlossenen Schritten $s^{E_i} \in \mathbf{S}^E$ zu den zusätzlichen Transitionen verlaufen. Diese sind in ihren Transitionsbedingungen identisch zu den Transitionen im Nachbereich des *einschließenden Schritts* $s_{E,m} \bullet$. Für alle *impliziten Wirkverbindungen*, die aus der Aktivierungsverbindung resultieren, gilt $c^{\varphi_{ij}} = 1$. In allen anderen Fällen gilt $c^{\varphi_{ij}} = 0$. Darüber hinaus gilt (6-20) für alle weiteren zusätzlichen Wirkverbindungen. Somit ergibt sich für die Inzidenzmatrix der Einschließungen aus Abbildung 6-6:

$$\mathbf{C}_\varphi = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (6-49)$$

Die Inzidenzmatrix des normalisierten Grafcet \mathbf{C}_N resultiert aus der Zusammenführung der Matrizen \mathbf{C} und \mathbf{C}_φ . Für das Beispiel aus Abbildung 6-6 ergibt sich somit:

$$\mathbf{C}_N = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & -1 \end{bmatrix} \quad (6-50)$$

\mathbf{C}_N kann nun in Verbindung mit (6-19) dazu verwendet werden, die erreichbaren Situationen des Grafcet unter Berücksichtigung des Prinzips der Einschließung schrittweise und unter Beachtung von (6-24) zu berechnen. Die Ausgabefunktion gemäß (6-25) bzw. (6-39) bleibt unverändert.

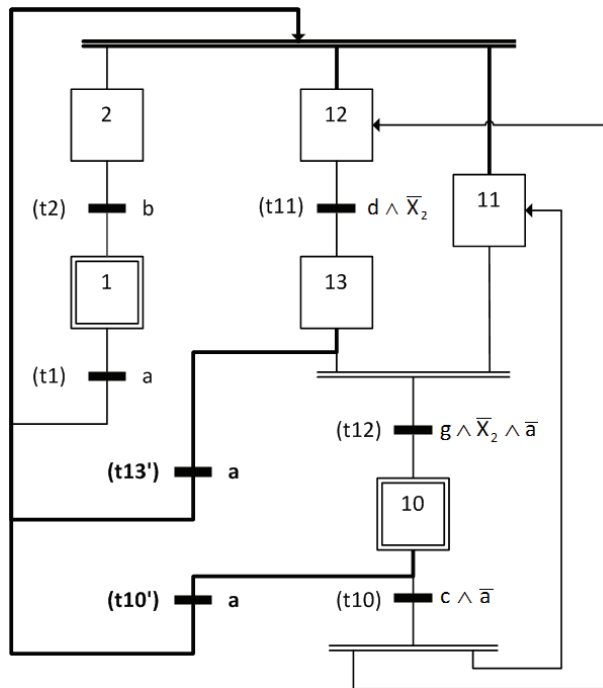


Abbildung 6-7: Normalform der Zwangssteuerung aus Abbildung 4-6

6.2.3 Zwangssteuernde Befehle

Durch *zwangssteuernde Befehle* können ebenfalls hierarchische Abhängigkeiten zwischen Teil-Grafcets spezifiziert werden. Eine Zwangssteuerung, bestehend aus einem *zwangssteuernden Befehl* und einem *zwangsgesteuerten Teil-Grafcet*, ist ein GRAFCET-interner Befehl und wirkt sich insofern auf den untergeordneten Teil-Grafcet aus, als dass dieser in einer bestimmten Situation eingefroren wird (\rightarrow 4.3.3). Erst wenn die Zwangssteuerung aufgehoben wird, unterliegt der zwangsgesteuerte Teil-Grafcet wieder den Ablaufregeln (\rightarrow 4.2.3). Wie bei *einschließenden Schritten*, werden die aus einer Zwangssteuerung resultierenden *impliziten Wirkverbindungen* auch im Falle von *zwangssteuernden Befehlen* durch die Transformation des hierarchisch strukturierten Grafcet in seine Normalform offensichtlich. So zeigt Abbildung 6-7 den normalisierten Grafcet G_N für das Beispiel der Zwangssteuerung aus Abbildung 4-6. Aus Gründen der Übersichtlichkeit wird auf die Darstellung von Aktionen, wie in Abbildung 6-6, verzichtet.

Der *zwangssteuernde Befehl* $G_{10}\{11, 12\}$ ist mit Schritt s_2 in Teil-Grafcet G_1 verknüpft und bewirkt das Einfrieren von G_{10} in der Situation

$$sit^Z(m) = \{X_{10}(m), X_{11}(m), X_{12}(m), X_{13}(m)\} = \{0, 1, 1, 0\},$$

sobald s_2 aktiviert wird und solange gilt $X_2 = 1$. Ungeachtet der vorangegangenen Situation zum Zeitpunkt $m - 1$ werden mit Aktivierung von s_2 nur diejenigen Schritte $S^{Z*} = \{s_{11}, s_{12}\} \subseteq S^Z$ des zwangsgesteuerten Teil-Grafcet $G^Z \cong G_{10}$ aktiviert, die durch den *zwangssteuernden Befehl* angesprochen werden. Alle anderen Schritte aus S^Z werden deaktiviert.

Die Transitionsbedingungen von G^Z müssen im Zuge der Normalisierung ebenfalls erweitert werden. So muss für jede Transition t^{Z*} im Nachbereich eines zwangsgesteuerten Schrittes aus S^{Z*} die entsprechende Transitionsbedingung durch die negierte Schrittvariable des mit dem *zwangssteuernden Befehl* assoziierten Schrittes \bar{X}_Z im Sinne einer Konjunktion erweitert werden. Für alle übrigen Transitionen $t \in G^Z$ muss die zugehörige Transitionsbedingung $r(t)$ durch die negierte Transitionsbedingung der Transitionen aus $s_Z \bullet$ im Sinne einer Konjunktion erweitert werden. Die Transitionsbedingungen mehrerer Transitionen in $s_Z \bullet$ werden durch Disjunktion miteinander verknüpft.

Für das Beispiel in Abbildung 6-7 muss für die Transitionsbedingungen $r(t_{10})$, $r(t_{11})$ und $r(t_{12})$ folglich gelten:

$$r(t_{10}) = c \wedge \bar{a} \quad (6-51)$$

$$r(t_{11}) = d \wedge \bar{X}_2 \quad (6-52)$$

$$r(t_{12}) = g \wedge \bar{X}_2 \wedge \bar{a} \quad (6-53)$$

Hinsichtlich des formalen SIPN-Modells für GRAFCET ergeben sich durch *zwangssteuernde Befehle* keine Auswirkungen auf die bisherigen Definitionen bezüglich der GRAFCET-Struktur. Da es sich bei einem *zwangssteuernden Befehl* a_z um eine besondere Form einer *kontinuierlich wirkenden Aktion* handelt, sind entsprechende Anpassungen der Definitionen bezüglich des Wirkungsteils von GRAFCET notwendig. Formel (6-13) wird demnach erweitert zu:

$$\mathbf{a} = \mathbf{a}_{\text{kont}} \cup \mathbf{a}_{\text{gesp}} \cup \mathbf{a}_z \quad (6-54)$$

$$\text{mit } \mathbf{a}_{\text{kont}} \cap \mathbf{a}_{\text{gesp}} \cap \mathbf{a}_z = \emptyset \quad (6-55)$$

Die Menge der *zwangssteuernden Befehle* \mathbf{a}_z eines Grafcet ist Teil der Menge der Aktionen, wobei gilt:

$$\mathbf{a}_z = \{a_{z,1}, \dots, a_{z,h}\} \quad (6-56)$$

mit h: Anzahl der *zwangssteuernden Befehle*

Ein *zwangssteuernder Befehl* $a_z(s_i) \in \mathbf{a}_z$ ist eine Boolesche Funktion, $f: \{0,1\}^{\mathbf{X}^Z} \rightarrow \{0,1\}$, die allen Schritten aus \mathbf{S}^{Z^*} den aktiven und allen übrigen Schritten $\mathbf{S}^{Z'} \in \mathbf{S}^Z$ den inaktiven Schrittzustand zuweist, solange der assoziierte Schritt s_i aktiv ist ($X_i = 1$):

$$a_{z,h}(s_i): \begin{cases} 1 \Leftrightarrow ((f: \{0,1\}^{\mathbf{X}^{Z^*}} \rightarrow \{1\}) \wedge (f: \{0,1\}^{\mathbf{X}^{Z'}} \rightarrow \{0\})), \text{ wenn } (X_i = 1) \\ 0, \text{ sonst} \end{cases} \quad (6-57)$$

Wird s_i deaktiviert, so wird auch der *zwangssteuernde Befehl* deaktiviert. Solange der mit dem *zwangssteuernden Befehl* assoziierte Schritt s_i aktiv ist, gilt G^Z als eingefroren. Das dynamische Verhalten eines Grafcet wird durch *zwangssteuernde Befehle* somit entscheidend beeinflusst, so dass eine Transition nur dann auslöst, wenn sie freigegeben und die zugehörige Transitionsbedingung erfüllt ist und keiner der Schritte in ihrem Vorbereich einer Zwangssteuerung unterliegt.

Folglich wird (6-22) erweitert zu:

$$\delta_j(m) \begin{cases} 1 \Leftrightarrow ((\forall s_i \in \bullet t_j : X_i = 1 \wedge a_z(s_i) = 0) \wedge ([r(t_j)] = 1)) \\ 0 \Leftrightarrow \text{sonst} \end{cases} \quad (6-58)$$

Weitere Auswirkungen bezüglich des dynamischen Verhaltens spiegeln sich in der Inzidenzmatrix des normalisierten Grafcet \mathbf{C}_N wider. Für \mathbf{C}_N ergibt sich in Anlehnung an (6-47):

$$\mathbf{C}_N = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,10} & c_{1,11} & c_{1,12} & c_{1,10'} & c_{1,13'} \\ c_{2,1} & c_{2,2} & c_{2,10} & c_{2,11} & c_{2,12} & c_{2,10'} & c_{2,13'} \\ c_{10,1} & c_{10,2} & c_{10,10} & c_{10,11} & c_{10,12} & c_{10,10'} & c_{10,13'} \\ c_{11,1} & c_{11,2} & c_{11,10} & c_{11,11} & c_{11,12} & c_{11,10'} & c_{11,13'} \\ c_{12,1} & c_{12,2} & c_{12,10} & c_{12,11} & c_{12,12} & c_{12,10'} & c_{12,13'} \\ c_{13,1} & c_{13,2} & c_{13,10} & c_{13,11} & c_{13,12} & c_{13,10'} & c_{13,13'} \end{bmatrix} \quad (6-59)$$

Die durch die Normalisierung resultierenden zusätzlichen Wirkverbindungen und Transitionen (\rightarrow Abbildung 6-7) lassen sich in Form einer Inzidenzmatrix der Zwangssteuerungen \mathbf{C}_Z zusammenfassen. Für die Werte der Elemente c_{ij}^Z gelten die folgenden Bedingungen für die *impliziten Wirkverbindungen*:

$$c_{ij}^Z = \begin{cases} -1 \Leftrightarrow \forall (s_{i'}^{Z'}, t_j) \in \mathbf{Post} | ((t_j \in s_{z,h} \bullet) \wedge (s_{i'}^{Z'} \in \mathbf{S}^{Z'} \subseteq \mathbf{S}^Z)) \\ 1 \Leftrightarrow \exists (t_j, s_{i'}^{Z*}) \in \mathbf{Post} | ((t_j \in \bullet s_{z,h}) \wedge (s_{i'}^{Z*} \in \mathbf{S}^{Z*} \subseteq \mathbf{S}^Z)) \\ 0, \text{ sonst} \end{cases} \quad (6-60)$$

Es gilt $c_{ij}^Z = -1$, für alle Postkanten, die als *implizite Wirkverbindungen* von den durch die Zwangssteuerung deaktivierten Schritten $s_{i'}^{Z'} \in \mathbf{S}^{Z'}$ zu den Transitionen im Vorbereich des Schrittes s_z , der mit dem *zwangssteuernden Befehl* verknüpft ist, verlaufen. Für alle *impliziten Wirkverbindungen*, die von den Transitionen im Vorbereich von s_z zu den durch die Zwangssteuerung aktivierten Schritten $s_{i'}^{Z*} \in \mathbf{S}^{Z*}$ verlaufen, gilt $c_{ij}^Z = 1$. In allen anderen Fällen gilt $c_{ij}^Z = 0$. Darüber hinaus gilt (6-20) für alle weiteren zusätzlichen Wirkverbindungen. Somit ergibt sich für die Inzidenzmatrix der Zwangssteuerungen aus Abbildung 6-7:

$$\mathbf{C}_Z = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (6-61)$$

und für die Inzidenzmatrix des normalisierten Grafcet \mathbf{C}_N :

$$\mathbf{C}_N = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 1 & 1 \\ 1 & 0 & 1 & -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & -1 \end{bmatrix} \quad (6-62)$$

\mathbf{C}_N kann nun in Verbindung mit (6-19) dazu verwendet werden, die erreichbaren Situationen des Grafnet unter Berücksichtigung des Prinzips der Zwangssteuerung schrittweise und unter Beachtung von (6-24) zu berechnen.

Die Ausgabefunktion gemäß (6-39) muss um die *zwangssteuernden Befehle* erweitert werden, für die die gleichen Ausführungsbedingungen gelten, wie für *kontinuierlich wirkende Aktionen*:

$$\Omega(m+1) =$$

$$\prod_{i=1, \substack{/S/ \\ \{\uparrow, \downarrow\}^X}} a_{\text{gesp}, j}(S_i) \wedge \prod_{j=1}^q a_{\text{gesp}, j}(S_i) \wedge \prod_{i=1, \substack{/S/ \\ X_i(m+1)=1 \wedge \\ \text{stabil}}} a_{\text{kont}, j}(S_i) \wedge \prod_{j=1, \substack{p \\ \text{Cond}_{\text{bool}} \\ (a_{\text{kont}, p}(S_i))=1}} a_{\text{kont}, j}(S_i) \wedge \prod_{i=1, \substack{/S/ \\ X_i(m+1)=1 \wedge \\ \text{stabil}}} a_{z, j}(S_i) \wedge \prod_{j=1}^h a_{z, j}(S_i) \quad (6-63)$$

6.2.4 Zeitabhängige Bedingungen

Die bisherigen Ausführungen zum formalen SIPN-Modell für GRAFCET berücksichtigen zeitliche Zusammenhänge lediglich indirekt durch die ereignisdiskrete Dynamik, die zu einer Abfolge von Situationen führt. Darüber hinaus definiert die IEC 60848 *zeitabhängige Bedingungen* (\rightarrow 4.4), die in Form von zeitabhängigen Transitionsbedingungen und *kontinuierlich wirkenden Aktionen* eine direkte Spezifikation zeitgebundener Anforderungen an den Steuerungsablauf erlauben. Sie erweitern das Spektrum logischer Aussagen, die mit Transitionsbedingungen und *kontinuierlich wirkenden Aktionen* verknüpft werden können.

Eine *zeitabhängige Bedingung* τ ist eine Funktion, $f: \{[t_1, t_2]\}^{\mathbb{R}} \rightarrow \{0, 1\}^X$, die einer Eingabe aus \mathbf{I} oder einer Schrittvariablen aus \mathbf{X} (zeitabhängige Variable) maximal zwei Verzögerungszeiten t_1 und t_2 zuordnet, deren Wertebereiche sich über die Menge der reellen Zahlen \mathbb{R} erstrecken. τ ist Bestandteil der Transitionsbedingungen und der Zuweisungsbedingungen für *kontinuierlich wirkende Aktionen* und entspricht den syntaktischen Vorgaben aus (4-1), (4-3) oder (4-4). Der Wert von τ ist abhängig vom Zustand der zeitabhängigen Variablen α und der Verzögerungszeiten t_1 und t_2 :

$$\tau: \begin{cases} 1, [t_1 / \uparrow \alpha, \dots, t_2 / \downarrow \alpha] \\ 0, \text{sonst} \end{cases} \quad (6-64)$$

Die *zeitabhängige Bedingung* ist erstmalig t_1 Zeiteinheiten nach dem Ereignis $\uparrow \alpha$ erfüllt und bleibt erfüllt bis t_2 Zeiteinheiten nach dem Ereignis $\downarrow \alpha$. In allen anderen Fällen ist τ nicht erfüllt.

Als Erweiterung von (6-10) setzt sich eine Transitionsbedingung $r(t_j) \in \mathbf{r}$ nunmehr zusammen aus einem Ereignis E_j , einer Bedingung B_j und einer *zeitabhängigen Bedingung* τ_j :

$$r(t_j) = E_j \wedge B_j \wedge \tau_j, j = 1..T/ \quad (6-65)$$

wobei für E_j und B_j (6-11) und (6-12) entsprechend gelten. Für jede Transition $t_j \in \mathbf{T}$ existiert eine entsprechende Transitionsbedingung $r(t_j)$, die nicht konstant *false* sein darf, wobei die immer erfüllte Bedingung durch $B_j = \text{true}$, das immer eintretende Ereignis durch $E_j = \varepsilon$ und die immer erfüllte *zeitabhängige Bedingung* durch $\tau_j = \text{true}$ festgelegt sind. Dies bedeutet,

dass die der Transitionsbedingung $r(t)$ zugeordnete logische Variable $[r(t)]$ entweder den Wert $[r(t)] = 1$ oder den Wert $[r(t)] = 0$ annehmen oder immer erfüllt sein kann. Es gilt $[r(t_j)] = 1$, wenn die Bedingungen B_j und τ_j erfüllt sind und das Ereignis E_j eintritt.

Für *kontinuierlich wirkende Aktionen* ergibt sich eine Erweiterung von (6-37), so dass gilt:

$$a_{\text{kont},p}(s_i): \begin{cases} 1, \text{ wenn } (X_i = 1) \wedge (Cond_{\text{bool},p} = 1) \wedge (\tau_p = 1) \\ 0, \text{ sonst} \end{cases} \quad (6-66)$$

Aus (6-65) und (6-66) folgt, dass durch die Berücksichtigung *zeitabhängiger Bedingungen* die momentane Situation eines Grafcet zu einem bestimmten Zeitpunkt nicht mehr nur abhängig vom Zustand der Schrittvariablen, sondern auch von der jeweils abgelaufenen Zeit innerhalb der *zeitabhängigen Bedingungen* ist. Die in (6-64) definierten Intervallgrenzen für $\tau = 1$ bedingen eine endliche Menge von Stoppuhren Θ , die jeder zeitabhängigen Variablen α zugeordnet sind und das Tupel des formalen SIPN-Modells für GRAFCET erweitern:

$$\Theta = \{\Theta(\alpha_1), \dots, \Theta(\alpha_k)\} \quad (6-67)$$

mit k : Anzahl der zeitabhängigen Variablen und $\alpha_k \in I \cup X$

Eine Stoppuhr $\Theta(\alpha_k)$ wird bei Eintreten des Ereignisses $\uparrow \alpha_k$ auf den Wert Null zurückgesetzt und gestartet. Das Erreichen des Wertes $\Theta(\alpha_k) = t_1$ ist gleichzusetzen mit dem Erreichen der linken Intervallgrenze für $\tau = 1$, so dass die *zeitabhängige Bedingung* erfüllt ist und $\Theta(\alpha_k)$ gestoppt wird. Tritt im weiteren Verlauf das Ereignis $\downarrow \alpha_k$ ein, so wird $\Theta(\alpha_k)$ auf den Wert Null zurückgesetzt und erneut gestartet. Bei Erreichen von $\Theta(\alpha_k) = t_2$ ist die rechte Intervallgrenze für $\tau = 1$ erreicht, was zu $\tau = 0$ und dem Stoppen von $\Theta(\alpha_k)$ führt.

Die momentane Situation *sit*(m) eines Basic-Grafcet zum Zeitpunkt m ist demzufolge durch die Menge der zum Zeitpunkt m aktiven Schritte und die Werte der Stoppuhren aus $\Theta(\alpha)$ charakterisiert, so dass in Erweiterung von (6-7) gilt:

$$\mathit{sit}(m) = \{X_1(m), \dots, X_{/S}(m), \Theta(\alpha_1), \dots, \Theta(\alpha_k)\} \quad (6-68)$$

Die Werte der Stoppuhren zum Zeitpunkt der Initialisierung sind $\Theta_0(\alpha_k) = 0$

In den vorangegangenen Teil-Abschnitten wurde aufgezeigt, dass für Einschließungen und Zwangssteuerungen eine Rückführung auf das formale Modell des Basic-Grafcet möglich ist. Ähnlich zu *einschließenden Schritten* und *zwangssteuernden Befehlen* besteht auch hinsichtlich der formal definierten Stoppuhren die Möglichkeit einer Normalisierung, mit Hilfe derer GRAFCET-spezifische zeitabhängige Konstrukte auf ein bewährtes formales Petrinetzmodell zurückgeführt werden können. Wie im Rahmen einer ausführlichen Untersuchung zeitbehafteter Petrinetze in [SCHUMACHER & FAY 2013 A]* herausgestellt wird, liefern die in [RAMCHANDANI 1974] definierten *T-timed Petrinetze* diesbezüglich einen Ansatzpunkt für eine Petrinetz-basierte Interpretation von *zeitabhängigen Bedingungen* in GRAFCET. Wie in Abbildung 6-8 veranschaulicht, kann jede Transition in einem *T-timed Petrinetz* mit einer Verzögerungszeit assoziiert sein. In dem gegebenen einfachen Beispiel der Abbildung muss die Transition T_1 genau d_1 Zeiteinheiten nachdem eine Marke die Stelle P_1 erreicht hat schalten, im Sinne einer schwachen Konzessionsregel. Wird die nach

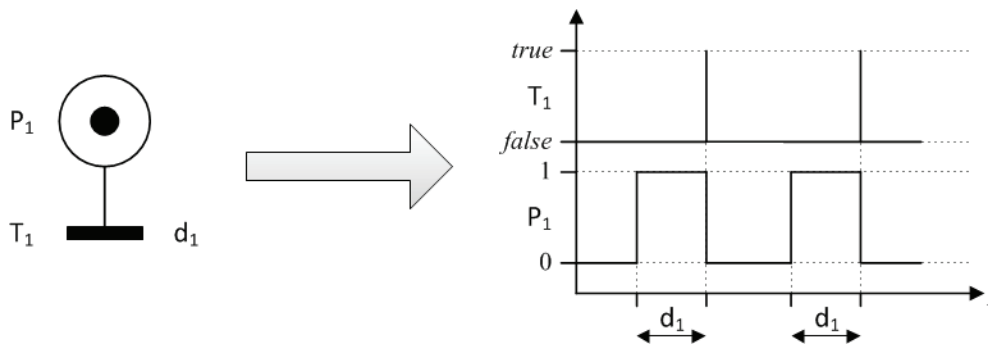


Abbildung 6-8: Dynamisches Verhalten eines T-timed Petrinetzes

dem Schalten von T_1 nicht mehr belegte Stelle P_1 anschließend erneut mit einer Marke belegt, so startet die Verzögerungszeit erneut und die Marke wird d_1 Zeiteinheiten später entzogen.

Gemäß ihrer eigentlichen Definition basieren *T-timed Petrinetze* auf den ST-Netzen [LITZ 2005]. Dementsprechend können die einzelnen Stellen eine ganzzahlige Menge von Marken enthalten. Ist für das dynamische Verhalten eines *T-timed Petrinetz* allerdings sichergestellt, dass jede Stelle zu keinem Zeitpunkt mehr als eine Marke enthalten kann, so entspricht die jeder Transition zugewiesene Verzögerungszeit d_j , aus theoretischer Sicht, in GRAFCET der *zeitabhängigen Bedingung* $d_j / *$, siehe Abschnitt 4.4 (4-3). Die zeitabhängige Variable wird in diesem Fall ersetzt durch die Schrittvariable des Vorgängerschritts. *Zeitabhängige Bedingungen* in *T-timed Petrinetzen* und GRAFCET zeigen, unter diesen Voraussetzungen, äquivalentes dynamisches Verhalten. Folglich können *zeitabhängige Bedingungen* in GRAFCET einem formalen Petrinetz-basierten Modell (Normalform) zugänglich gemacht werden, wenn alle Konstrukte *zeitabhängiger Bedingungen* gemäß IEC 60848 auf zeitabhängige Transitionsbedingungen entsprechend (4-3) zurückgeführt werden können. Im Gegensatz zu hierarchischen Konstrukten, bei deren Normalisierung *implizite Wirkverbindungen* offenkundig werden, müssen im Falle der Normalisierung *zeitabhängiger Bedingungen* nicht nur Transitionen und Wirkverbindungen im Grafcet ergänzt werden. Vielmehr resultiert aus der Normalisierung einer Stoppuhr, welche einer zeitabhängigen Variablen zugeordnet ist, ein zusätzlicher Teil eines Grafkets, welcher das Stoppuhrverhalten nachbildet und mit dem ursprünglichen Grafcet verkoppelt ist.

Abbildung 6-9 zeigt dazu den allgemeinen Fall einer zeitabhängigen Transitionsbedingung am Beispiel eines GRAFCET-Ausschnitts und den zugehörigen normalisierten Grafcet. Die Kopplung der Stoppuhr an den ursprünglichen Grafcet erfolgt durch einen zusätzlichen Schritt, der hier mit dem Namen 1' versehen ist. Die vormals zeitabhängige Transitionsbedingung von t_1 wird durch die immer erfüllte Transitionsbedingung *true* ersetzt. Durch die erweiterte Menge von Schritten und Transitionen muss Schritt s_{20} als Initialschritt deklariert werden, um äquivalentes dynamisches Verhalten zu gewährleisten. Der somit normalisierte Grafcet entspricht aus zeitlicher Hinsicht formal einem *T-timed Petrinetz*.

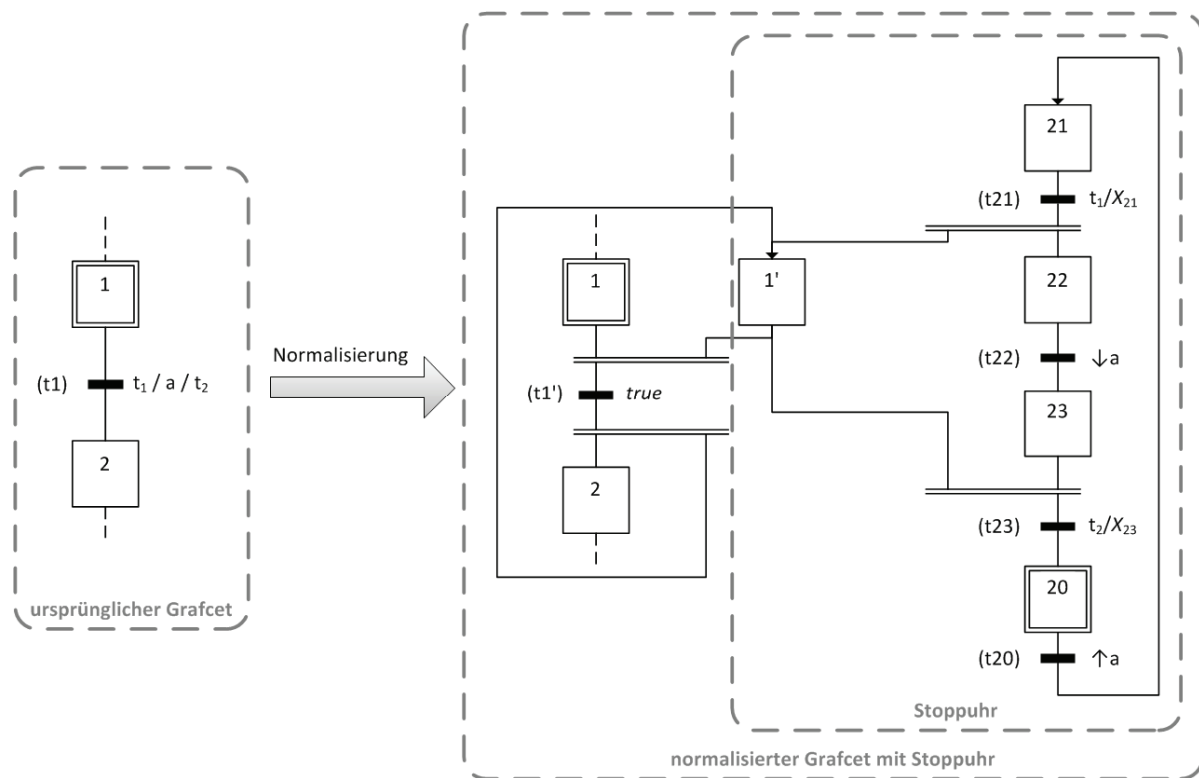


Abbildung 6-9: Normalisierung zeitabhängiger Transitionsbedingungen

Auch für den allgemeinen Fall zeitabhängiger *kontinuierlich wirkender Aktionen* kann das Stoppuhrverhalten im Rahmen der Normalisierung durch zusätzliche Schritte und Transitionen abgebildet werden (→ Abbildung 6-10). Aufgrund der spezifischen Eigenschaften von Zuweisungsbedingungen muss hier aber noch der Fall berücksichtigt werden, dass die *kontinuierlich wirkende Aktion* in Folge des Auslösens einer dem assoziierten Schritt nachfolgenden Transition in einer Situation nicht ausgeführt wird. Auch in diesem Fall entspricht der normalisierte Grafcet aus zeitlicher Hinsicht formal einem *T-timed Petrinetz*.

Die Normalisierung zeigt, dass die gemäß IEC 60848 möglichen *zeitabhängigen Bedingungen*, auch die in den allgemeinen Fällen enthaltenen abgekürzten Schreibweisen, formal auf zeitbewertete Petrinetze, speziell auf *T-timed Petrinetze* zurückgeführt werden können.

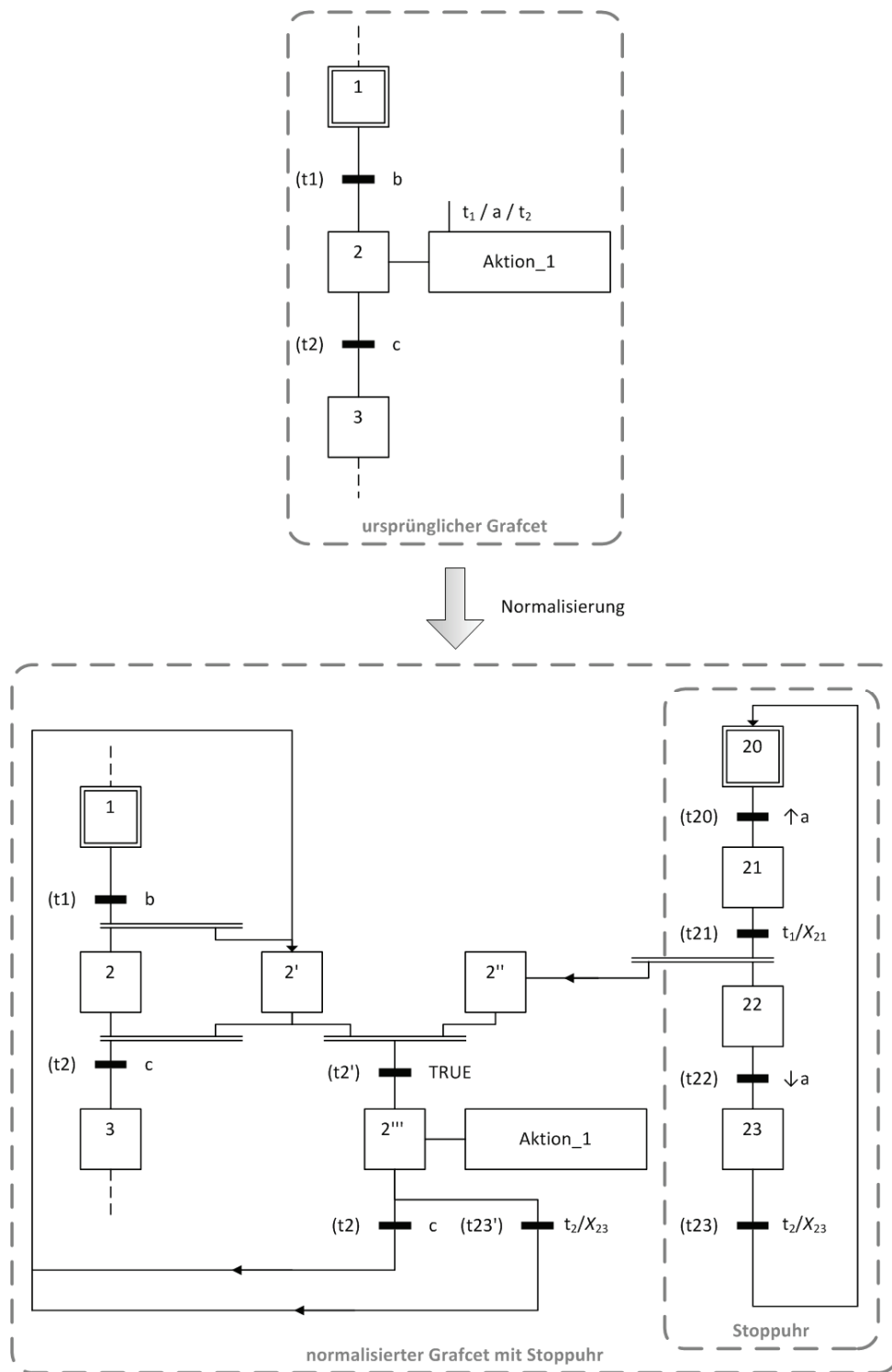


Abbildung 6-10: Normalisierung zeitabhängiger Aktionsbedingungen

6.4 Zwischenfazit

Das im Rahmen dieses Kapitels vorgestellte formale SIPN-Modell bietet eine umfassende und eindeutige Definition für die Struktur, den Wirkungsteil und das dynamische Verhalten von GRAFCET gemäß IEC 60848. Das formale Modell geht über bisherige Forschungsansätze hinaus und berücksichtigt neben Booleschen Zuweisungsbedingungen und Aktionen bei Ereignissen auch *zeitabhängige Bedingungen* und Elemente zur hierarchischen Strukturierung in Form *einschließender Schritte* und *zwangsteuernder Befehle*. Bis auf die *Aktion bei Auslösung* sind somit alle Elemente der IEC 60848 in das formale SIPN-Modell integriert, die das für GRAFCET wesentliche Prinzip, nach dem grundsätzlich Transitionsbedingungen mit Transitionen und Aktionen mit Schritten verknüpft werden, erfüllen.

Alle GRAFCET-Elemente, die über den Beschreibungsumfang eines Basic-Grafcet hinausgehen, werden dazu als Erweiterung des Basic-Grafcet angesehen. Insbesondere bei hierarchisch strukturierten Grafcets zeigt der Ansatz der Normalisierung, dass es sich bei *einschließenden Schritten* und *zwangsteuernden Befehlen* um kompakte Ausdrücke handelt, deren Verhalten auf einen Basic-Grafcet zurückgeführt werden kann. In Bezug auf die Spezifikation zeitabhängiger Anforderungen können bereits existierende Formalismen aus dem Bereich der zeitbewerteten Petrinetze nicht direkt für GRAFCET übernommen werden, da sowohl syntaktische Vorgaben als auch deren Interpretation verschieden sind. Dies zeigt beispielsweise eine Gegenüberstellung in [SCHUMACHER & FAY 2013 A]*. Durch die Rückführung auf T-timed Petrinetze können aber auch *zeitabhängige Bedingungen* gemäß IEC 60848 vollständig in das formale SIPN-Modell für GRAFCET integriert werden.

Nachdem das Beschreibungsmittel GRAFCET durch die formale Betrachtung als SIPN nun eindeutig definiert ist, gilt es im Folgenden, geeignete Methoden und Werkzeuge bereitzustellen, die einen systematischen Steuerungsentwurf unterstützen. Eine Grundvoraussetzung dafür ist, die Informationen eines Grafcet in einem möglichst offenen, maschinenlesbaren Format abzulegen, so dass weiterführende Automatismen, wie beispielsweise zur formalen Verifikation, darauf zugreifen können. Das nachfolgende Kapitel widmet sich aus diesem Grund einer implementierungsunabhängigen Notation für GRAFCET.

7 Implementierungsunabhängige Notation für GRAFCET

7.1 Voraussetzungen für eine implementierungsunabhängige Notation

7.1.1 Anforderungen an eine implementierungsunabhängige Notation

Wie bereits in vorangehenden Kapiteln beschrieben (→ 4.5), nutzen momentan verfügbare Werkzeuge für GRAFCET nicht-offene, proprietäre Datenformate. Die Informationen bezüglich der Spezifikation eines Steuerungsablaufs werden als Daten so gespeichert, dass andere Werkzeuge nicht direkt darauf zugreifen können, und verhindern dadurch die Adaption weiterführender Funktionalitäten im Sinne eines systematischen Steuerungsentwurfes. Folglich ist eine wesentliche Voraussetzung, um den systematischen Steuerungsentwurf mit GRAFCET durch geeignete Methoden und Werkzeuge zu unterstützen, ein offenes maschinenlesbares Format, in dem GRAFCET-spezifische Daten abgelegt werden können. Eine solche implementierungsunabhängige Notation dient dem Zweck, den Datenaustausch zwischen Engineering-Werkzeugen zu ermöglichen, und ist zentrale Datendrehscheibe für den systematischen, werkzeugunterstützten Steuerungsentwurf mit GRAFCET.

Eine mit Hilfe eines GRAFCET-Editors erstellte Spezifikation kann entweder direkt oder über eine zusätzliche Transformation in die geeignete implementierungsunabhängige Notation überführt werden. Sollte eine zusätzliche Transformation notwendig sein, so muss die werkzeugspezifische Datenrepräsentation des GRAFCET-Editors offen zugänglich sein, um eindeutige Abbildungsregeln definieren zu können. Die gemäß den Vorgaben der implementierungsunabhängigen Notation abgelegten GRAFCET-Daten stehen anschließend einer weiterführenden Be- und Verarbeitung zur Verfügung, wie beispielsweise zur Verifikation durch Algorithmen oder zur automatischen Generierung von Steuerungscode. Auch das Einlesen des Datenformats in den GRAFCET-Editor oder andere Editoren ist denkbar, sofern entsprechende Abbildungsregeln definiert worden sind. Der Fokus dieser Arbeit liegt darauf, eine Methode für die algorithmengestützte Transformation von GRAFCET-Daten in eine implementierungsunabhängige Notation zu entwickeln und durch ein geeignetes Werkzeug zu unterstützen.

Die *eXtensible Markup Language* (XML) [W3C XML 1.0][#] bietet hierzu ein offenes, textbasiertes Format zur strukturierten Ablage von Daten und eine breite werkzeugtechnische Unterstützung. XML wurde ursprünglich für den Bereich informationstechnischer Anwendungen durch das *World Wide Web Consortium* (W3C) als offener Standard entwickelt und vorangetrieben [W3C][@]. Als Beschreibungssprache für Daten (Metasprache) ermöglicht XML, Daten losgelöst von deren Anwendung und in einer definierten Struktur textuell abzuspeichern und erleichtert somit einen Datenaustausch. Auch die zugehörigen sprachlichen Erweiterungen, wie beispielsweise zur Definition anwendungsspezifischer XML-Strukturen [W3C XSD 1.1 PART 1, W3C XSD 1.1 PART 2][#] oder zur Transformation von XML-Dokumenten in andere XML-Dokumente [W3C XSLT 2.0][#], werden durch Arbeitsgruppen des W3C definiert und verwaltet und bieten somit ein offen zugängliches Sprachnetzwerk zur einfachen Erstellung und Handhabung von XML-Dateien. In den letzten Jahren haben diese XML-Technologien auch vermehrt Einzug in automatisierungstechnische Anwendungen gefunden und zählen mittlerweile auch in diesem Bereich zum Stand der Technik [WOLLSCHLAEGER & WENZEL 2005]. So existiert eine Vielzahl spezifischer XML-

Strukturen (XML-Schemata), die jeweils bestimmte Anwendungsfälle im automatisierungstechnischen Umfeld abdecken, welches nach [WOLLSCHLAEGER ET AL. 2010] als positive und gleichzeitig negative Eigenschaft von XML charakterisiert wird. Eine fehlende methodische Vorgehensweise zur Erstellung XML-basierter Datenaustauschformate unter der Berücksichtigung bereits bestehender Lösungen veranlasste daher den Fachausschuss 5.23 „XML in der Automation“ der *Gesellschaft für Mess- und Automatisierungstechnik* im VDI/VDE (GMA) [GMA][@], eine Richtlinie zu erarbeiten, die ein entsprechendes Vorgehensmodell für den XML-Anwender bereitstellt [WOLLSCHLAEGER ET AL. 2010]. Das Vorgehensmodell der VDI/VDE-Richtlinie 3690 [VDI/VDE 3690, BLATT 2][#], die aus insgesamt drei Richtlinienblättern besteht, soll im Rahmen dieser Arbeit angewendet werden, um die Daten eines Grafcet systematisch in ein XML-Datenformat zu überführen. Sofern möglich, sollten für die implementierungsunabhängige Notation von GRAFCET ein bereits etabliertes offenes Datenformat und die zugehörigen Konventionen genutzt werden, um einerseits den Entwicklungsaufwand möglichst niedrig zu halten und andererseits mit diesem Datenformat verknüpfte Werkzeuge nutzen zu können. Abbildung 7-1 zeigt hierzu die wesentlichen Schritte, die auf dem Weg zu einem XML-basierten Datenaustauschformat für GRAFCET und in Anlehnung an die VDI/VDE-Richtlinie 3690 berücksichtigt werden müssen.

Durch die Analyse existierender XML-Schemata wird in einem ersten Schritt geprüft, welcher Umfang an GRAFCET-Elementen eindeutig in der jeweiligen XML-Struktur abgebildet werden kann. Der resultierende Abdeckungsgrad ist entscheidend für die grundsätzliche Eignung oder Nichteignung des Datenaustauschformats. Es sollten auch komplexe Grafkets, beispielsweise in einer hierarchischen Struktur verknüpfte Teil-Grafkets, im XML-Schema abgebildet werden können und die Namensgebung der XML-Elemente nach Möglichkeit mit der Namensgebung der GRAFCET-Elemente übereinstimmen. Der Grafcet muss in diesem Zusammenhang nicht zwangsläufig vollständig beschrieben sein, so dass auch unvollständige Grafkets oder ausgewählte Teile einer Spezifikation im Datenaustauschformat abgelegt werden können. Werkzeugspezifische Daten sollten als solche besonders gekennzeichnet werden. Der wesentliche Inhalt der Spezifikation sollte aber für alle Werkzeuge lesbar sein. Des Weiteren sollten grafische Informationen über die GRAFCET-Elemente und deren Anordnung optional gespeichert werden können. Der Beschreibung grafischer Information sollte allerdings keine Priorität beigemessen werden, so dass die funktionalen Zusammenhänge eines Grafcet auch hinsichtlich des Datenaustauschformats im Vordergrund stehen.

Anhand dieser spezifischen Anforderungen und unter Beachtung der zuvor beschriebenen allgemeinen Anforderungen an eine implementierungsunabhängige Notation für GRAFCET können ein geeignetes XML-Schema identifiziert und eindeutige Abbildungsregeln in einem zweiten Schritt definiert werden. Auf der Grundlage eines Vergleichs etablierter Auszeichnungssprachen (→ 7.1.2) werden anschließend das in dieser Arbeit verfolgte

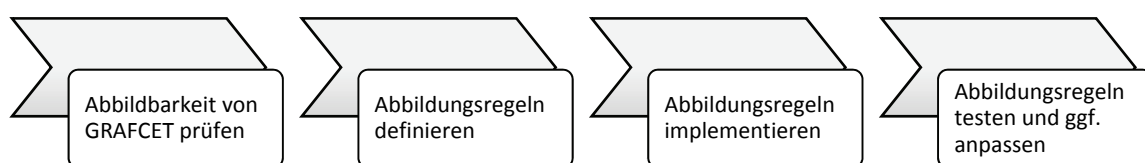


Abbildung 7-1: Vorgehensmodell in Anlehnung an [VDI/VDE 3690, BLATT 2][#]

Konzept für eine implementierungsunabhängige Notation für GRAFCET (→ 7.2) und dessen Umsetzung (→ 7.3) erläutert. Die gemäß Abbildung 7-1 anschließenden Schritte werden anhand einer prototypischen Implementierung im weiteren Verlauf dieser Arbeit behandelt (→ 9.2). Einen umfassenden Überblick über derzeit etablierte XML-basierte Auszeichnungssprachen im Kontext der Automatisierung gibt [VDI/VDE 3690, BLATT 1][#]. Neben einer Klassifizierung anhand der Lebenszyklusphasen eines automatisierten technischen Systems bietet Blatt 1 der VDI/VDE-Richtlinie 3690 ebenfalls eine tabellarische Übersicht über die funktionale Eignung der aufgeführten Auszeichnungssprachen. Die Kategorie **Steuerungsfunktionen** umfasst XML-Schemata, in denen Funktionen zur „*Informationsverarbeitung (Steuern und Regeln) und zum lokalen Bedienen und Beobachten*“ [VDI/VDE 3690, BLATT 1, KAPITEL 5.1][#] abgelegt werden können. Die Auszeichnungssprachen *PROFINET Component Description* (PCD), *Substation Configuration Description Language* (SCL), *BatchML*, *PLCopenXML* und *AutomationML* werden hierbei als diejenigen XML-Schemata eingestuft, deren hauptsächliche Verwendung in der Kategorie **Steuerungsfunktionen** liegen, und bilden somit potentielle implementierungsunabhängige Notationen für GRAFCET, wie in Tabelle 7-1 dargestellt.

Tabelle 7-1: Auszug aus der funktionalen Klassifizierung von Auszeichnungssprachen gemäß [VDI/VDE 3690, BLATT 1][#]

XML-Lösung für:	Funktionale Klassifizierung				
	Feldfunktionen	Kommunikationsfunktionen	Steuerungsfunktionen	Leitfunktionen	MES- und ERP-Funktionen
domänenspezifische Anwendungen					
<i>BatchML</i> (IEC 61512 / ISA S S88)	O	-	X	X	O
PCD	O	X	X	-	-
<i>PLCopenXML</i>	O	-	X	O	-
SCL (IEC 61850-6)	X	X	X	O	-
domänenunabhängige Anwendungen					
<i>AutomationML</i>	O	-	X	X	-

Legende: X: Haupteinsatzbereich O: Einsatz möglich - : Einsatz nicht vorgesehen

Neben diesen wird auch die *Petri Net Markup Language* (PNML), aufgrund des formalen SIPN-Modells für GRAFCET (→ 6), in die Betrachtungen zu den etablierten Auszeichnungssprachen einbezogen, in dem diese den GRAFCET-spezifischen Anforderungen gegenübergestellt werden. Die in der VDI/VDE-Richtlinie 3690, Blatt 1, aufgeführte funktionale Klassifizierung berücksichtigt PNML nicht. Aufgrund der Vielfalt von Petrinetzen wäre die PNML grundsätzlich für jeden der aufgeführten Bereiche einsetzbar. Im Rahmen dieser Arbeit begrenzt sich der Einsatzbereich der PNML auf die Beschreibung von Steuerungsfunktionen gemäß Tabelle 7-1.

7.1.2 Vergleich etablierter Auszeichnungssprachen

7.1.2.1 *PROFINET Component Description*

Die PCD ist eine XML-basierte Auszeichnungssprache, welche für den Austausch von Gerätebeschreibungsdaten PROFINET-fähiger verteilter Automatisierungskomponenten zwischen Werkzeugen zur Parametrierung, Diagnose oder Kommunikationsnetzplanung vorgesehen ist [PROFINET 2006][#]. PCD-Spezifikation und das zugehörige XML-Schema sind für Mitglieder der *PROFIBUS/PROFINET* Nutzerorganisation frei verfügbar und in dem speziellen Anwendungsfall des Austauschs von Gerätebeschreibungsdaten nutzbar [PROFIBUS / PROFINET][@]. Die PCD-Struktur ist angelehnt an die in [ISO 15745-1][#] und [ISO 15745-3][#] definierte Struktur und bietet eine technologiespezifische Beschreibung von mechanischen, elektrotechnischen und informationstechnischen Geräteeigenschaften (*Component Based Automation*) der jeweiligen PROFINET-Komponente. Typischerweise handelt es sich bei einer solchen PROFINET CBA-Komponente um ein technologisches Modul, dessen Automatisierungsfunktionalität durch einen Funktionsblock [IEC 61499-1][#] spezifiziert wird. Gemäß PROFINET-CBA erfüllt jedes Modul in einem verteilten Automatisierungssystem eine technologische Funktion, die durch die PCD beschrieben wird. Sie ist zunächst unabhängig von den anderen technischen Modulen und beinhaltet Ein- und Ausgangsvariablen, Ein- und Ausgangsereignisse sowie die Beschreibung des Verhaltens in Form von Zuständen und Zustandsübergängen. Die Verhaltensbeschreibung beruht auf einem Zustandsautomaten-Modell, dessen dynamisches Verhalten durch einen Algorithmus konkretisiert wird. Insbesondere bei parallelen Abläufen gelangt die PCD dadurch an Grenzen, so dass das PCD-XML-Schema hinsichtlich einer implementierungsunabhängigen Notation für GRAFCET ungeeignet ist.

7.1.2.2 *Substation Configuration Description Language*

Die SCL gemäß IEC 61850-6 [IEC 61850-6] definiert ein XML-basiertes Datenaustauschformat für die Beschreibung von Kommunikationsnetzwerken im Bereich von Automatisierungssystemen zur Energieversorgung. Diese sogenannten Stationsautomatisierungssysteme können in SCL durch hierarchisch gegliederte Teilmodelle beschrieben werden, die durch unterschiedliche Kommunikationsverbindungen miteinander verknüpft sind. Neben der Beschreibung der Kommunikationsverbindungen innerhalb des Stationsautomatisierungssystems im sogenannten *Communication System Model* ermöglicht die SCL darüber hinaus sowohl eine Beschreibung der funktionalen hierarchischen Struktur (*Substation Model*) als auch eine hierarchische Perspektive auf die einzelnen Geräte (*Intelligent Electronic Device*) [MACKIEWICZ 2006]. Eine gültige SCL-Datei muss nicht zwingenderweise alle drei Modelle beinhalten. Sie dient dem XML-basierten Datenaustausch zwischen unterschiedlichen Engineering-Werkzeugen für Stationsautomatisierungssysteme. Funktionale Aspekte der einzelnen Geräte, wie beispielsweise Automatisierungsfunktionen, können in der SCL allerdings nur benannt und nicht im Detail spezifiziert werden. So ist etwa die Beschreibung des logischen Verhaltens der Geräte im Sinne einer Ablaufbeschreibung in SCL nicht vorgesehen. Hinsichtlich einer implementierungsunabhängigen Notation für GRAFCET ist die SCL daher ungeeignet.

7.1.2.4 *BatchML*

International in [IEC 61512-2][#] genormt, ist die **Batch Markup Language** (*BatchML*) ein integraler Bestandteil des ISA 88-Standards [ISA 88.02], welcher als weltweit akzeptierter Standard für Chargenprozesse gilt. Als XML-Schema stellt *BatchML* eine implementierungsunabhängige Notation für die Beschreibung von Rezepten und Chargenprozessen zur Verfügung und wird in der praktischen Anwendung durch kommerzielle, in der Prozessindustrie verbreitete Engineering-Werkzeuge unterstützt. Entsprechende werkzeugspezifische Import- und Exportschnittstellen ermöglichen somit einen einfachen Datenaustausch von Rezeptdaten auf der Basis von *BatchML*. Die aktuelle Version 4.01 für *BatchML* wurde in die **Business to Manufacturing Markup Language** (B2MML) integriert, die eine XML-basierte Implementierung der Modelle in [IEC 62264-1][#] darstellt. Der logische Ablauf eines Rezepts, wie beispielsweise eines Steuerrezepts, kann durch einen sogenannten Prozedur-Funktionsplan spezifiziert werden, welcher aus Schritten, Transitionen und Verbindungen besteht. Abbildung 7-2 zeigt dazu einen Ausschnitt der XML-Schema-Definition des Prozedur-Funktionsplans aus dem frei verfügbaren XML-Schema der Version 4.01.

Die Schrittketten des Prozedur-Funktionsplans, die auch alternative oder parallele Sequenzen enthalten können, besitzen eine große Ähnlichkeit mit der Struktur eines Grafocet. Grundsätzlich bietet sich somit die Möglichkeit, die Daten einer GRAFCET-Struktur entsprechend in *BatchML* abzulegen. Allerdings sind die einzelnen Rezepte, die als Master- oder Steuer-Rezepte in *BatchML* abgelegt werden können, gleichberechtigt und stehen in keiner direkten hierarchischen Beziehung zueinander. Eine zeitabhängige Koordination der einzelnen Rezepte erfolgt durch einen übergeordneten tabellarischen Chargenplan, dessen Inhalt im Element *BatchList* enthalten ist. Die Abbildung hierarchisch strukturierter Grafocets würde eine entsprechende Interpretation des dynamischen Verhaltens erfordern und kann durch eine einfache, pragmatische Abbildungsvorschrift, wie sie im Rahmen dieser Arbeit angestrebt wird, nicht geleistet werden. Hinsichtlich einer implementierungsunabhängigen Notation für GRAFCET wird *BatchML* somit als ungeeignet bewertet.

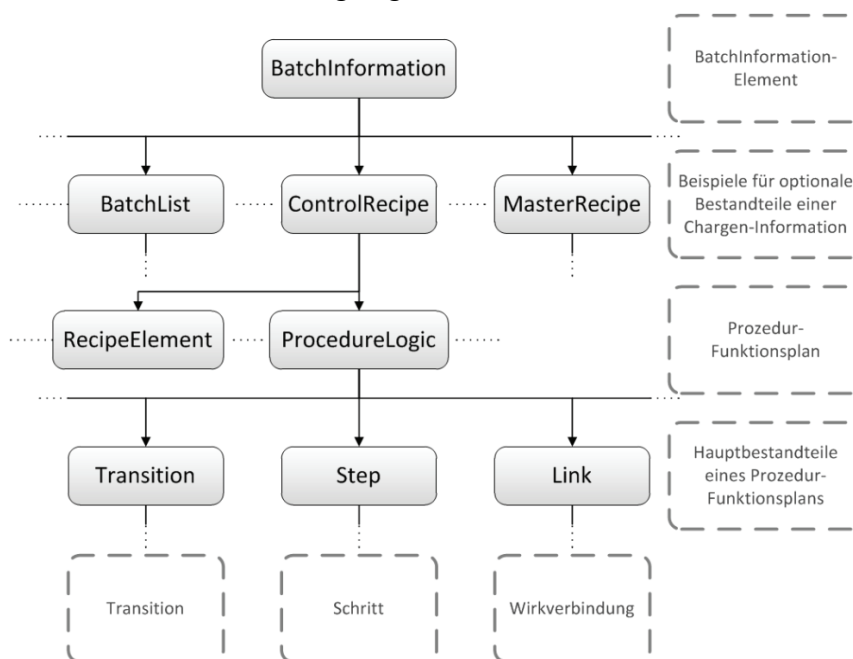


Abbildung 7-2: Strukturbaum der XML-Repräsentation von Prozedur-Funktionsplänen gemäß IEC 61512-2

7.1.2.5 *PLCopenXML*

PLCopenXML [PLCOOPEN 2009][#] ist ein offenes, XML-basiertes Datenaustauschformat für die Implementierung IEC 61131-3 konformer Steuerungsprogramme und wurde durch die Nutzerorganisation *PLCopen* [PLCOOPEN][@] erstmalig in der Version 1.01 im Jahre 2005 veröffentlicht. Das aktuelle XML-Schema liegt in der Version 2.0 aus dem Jahre 2008 vor. Ziel von *PLCopenXML* ist es, Steuerungsprojekte speicherprogrammierbarer Steuerungen in einem herstellerunabhängigen Datenformat abspeichern und zwischen unterschiedlichen SPS-Engineering-Werkzeugen austauschen zu können. Hierzu übernimmt *PLCopenXML* die durch IEC 61131 definierten Strukturen eines Steuerungsprojekts und berücksichtigt alle fünf Programmiersprachen der IEC 61131-3. Ein Steuerungsprojekt besteht grundsätzlich aus einem Element *instances*, in dem konkrete Konfigurationen von Steuerungsprogrammen abgespeichert werden, und aus dem Element *types*, dem Datentypen und POEs zugeordnet sind (→ Abbildung 7-3). Ein Steuerungsprogramm bildet das Grundgerüst einer Programmorganisationseinheit und kann durch die separate Angabe von Schnittstellen, Aktionen und Transitionen ergänzt werden.

Das *PLCopenXML*-Format ermöglicht die Abbildung aller in IEC 61131-3 definierten SFC-Elemente. Hinsichtlich einer implementierungsunabhängigen Notation für GRAFCET können die Gemeinsamkeiten von GRAFCET und SFC nützlich sein, um GRAFCET-Elemente direkt in *PLCopenXML* abzubilden, sofern beispielsweise ausschließlich Elemente eines Basic-Grafcet in Betracht gezogen werden. Hierarchisch strukturierte Grafcets müssten aber zuvor normalisiert werden, da sich die Möglichkeiten zur hierarchischen Strukturierung für GRAFCET und SFCs erheblich voneinander unterscheiden. Bei der Abbildung eines Grafcet in *PLCopenXML* ginge somit jegliche Information über Teil-Grafcets und ihre hierarchischen Abhängigkeiten verloren. Aus diesen Gründen wird *PLCopenXML* als potentielle implementierungsunabhängige Notation für GRAFCET nicht weiter betrachtet.

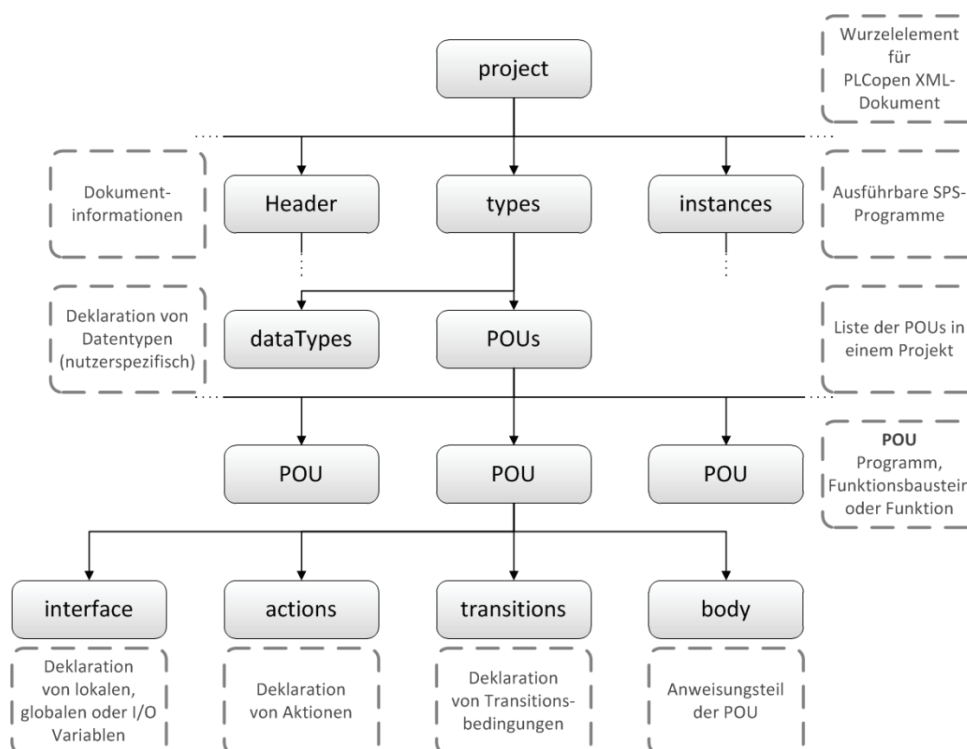


Abbildung 7-3: Strukturbaum von *PLCopenXML*

7.1.2.6 *AutomationML*

Um einen offenen Datenaustausch zwischen Engineering-Werkzeugen für die Automatisierungsplanung zu gewährleisten, begann im Jahre 2006 die Entwicklung des XML-basierten Datenaustauschformats *AutomationML*. Es ist über den Verein *AutomationML e.V.* frei zugänglich und befindet sich derzeit im internationalen Normungsprozess [65E/280/CDV][#]. *AutomationML* hat die Zielsetzung, zentrales Datenaustauschformat in einer durch Heterogenität geprägten Werkzeuglandschaft zu sein, und adaptiert hierfür die bereits etablierten offenen Datenaustauschformate CAEX [IEC 62424][#], *COLLADA* [COLLADA][@] und *PLCopenXML*. Die Anlagentopologie wird in CAEX abgebildet, welches gleichzeitig das Dachformat für *AutomationML* repräsentiert. Ausgehend von diesem Dachformat können in *COLLADA* gespeicherte geometrische und kinematische Modelle sowie in *PLCopenXML* gespeicherte Verhaltensmodelle verlinkt werden. Darüber hinaus können weiterführende semantische Definitionen in Form von Rollen- oder Schnittstellenbibliotheken in *AutomationML* integriert werden. Hinsichtlich einer implementierungsunabhängigen Notation ergibt sich somit für *AutomationML* die gleiche Bewertung wie für *PLCopenXML*.

7.1.2.7 *Petri Net Markup Language*

Die PNML ist ein auf XML basierendes Datenaustauschformat für Petrinetze und international in ISO/IEC 15909 standardisiert. Teil 1 des Standards [ISO/IEC 15909-1][#] beschreibt das Konzept, die grafische Repräsentation und das formale Modell für *High-level Petrinetze*, die als Beschreibungsmittel für ereignisdiskrete Systeme definiert werden. *High-level Petrinetze* bilden die Obermenge aller Petrinetze, so dass beispielsweise ST-Netze als Teilmenge von *High-level Petrinetzen* definiert werden können [HILLAH ET AL. 2009]. In Teil 2 des Standards [ISO/IEC 15909-2][#] wird das zugehörige Datenaustauschformat PNML konzeptionell und in Form einer XML-Syntax definiert. Die PNML hat den Anspruch, jede Art von Petrinetz durch das XML-Schema abbilden zu können. Der konzeptionelle Aufbau der PNML beruht aus diesem Grund auf einem Petrinetz-Kernmodell (*PNML Core Model*), welches Petrinetze allgemein und in abstrakter Form charakterisiert (→ Abbildung 7-4). Demnach ist ein Petrinetz ein bipartiter Digraph, bestehend aus den disjunkten Knotenmengen der Stellen und Transitionen, die über gerichtete Verbindungen (Kanten) miteinander verknüpft sind und auf einem Zeichenblatt grafisch abgebildet werden. Alle darüber hinausgehenden Eigenschaften dieser grundlegenden Petrinetz-Objekte, wie beispielsweise Kantengewichte oder Markierung der Stellen, sind abhängig vom jeweiligen Petrinetz-Typ und werden über sogenannte *Labels* mit den jeweiligen Petrinetz-Objekten assoziiert. Die Labels werden in *Annotationen* und *Attribute* unterteilt, wobei Annotationen Informationen der Petrinetz-Objekte repräsentieren, die in ihrer grafischen Erscheinung direkt sichtbar sind. Die einzige definierte Annotation des PNML Core Models ist der Name eines Petrinetz-Objekts. Attribute werden hingegen nicht grafisch abgebildet. Spezifische Eigenschaften des erstellenden Petrinetz-Werkzeugs können als solche eindeutig gekennzeichnet werden.

Somit bildet das PNML Core Model ein abstraktes Grundgerüst, auf dessen Basis eine typspezifische XML-Syntax definiert werden kann. Im normativen Teil der ISO/IEC 15909-2 werden PNML-Metamodelle für insgesamt drei Petrinetz-Typen definiert, ST-Netze, symmetrische Netze und *High-level Petrinetze*. Zwar besteht die Möglichkeit, aufbauend auf dem PNML Core Model weitere PNML-Metamodelle zu definieren, allerdings sind diese dann nicht mehr normkonform. Eine methodische Vorgehensweise, wie spezifische PNML-

Metamodelle systematisch und normkonform erstellt werden können, existiert momentan noch nicht, wird aber im Rahmen des in der Entstehung befindlichen dritten Teils des Standards erarbeitet [HILLAH & PETRUCCI 2010]. Weiterhin wird auf Seiten der Werkzeuge in [HILLAH ET AL. 2006] eine Referenzimplementierung diskutiert, die eine einfache Realisierung werkzeugspezifischer Import- und Exportmechanismen und somit eine einfache Kopplung an bereits vorhandene Petrinetz-Werkzeuge erlaubt. Hinsichtlich einer implementierungsunabhängigen Notation für GRAFCET deckt der Abstraktionsgrad der PNML die Anforderungen vollständig ab. Die zu einem hierarchisch strukturierten Grafcet zugehörigen Teil-Grafcets können auf verschiedenen Seiten des PNML-Datenformats abgelegt und die zugehörigen Verknüpfungen durch entsprechende Labels spezifiziert werden. In gleicher Art und Weise können Aktionen und Transitionsbedingungen zugeordnet werden, wobei die Namensgebung der einzelnen XML-Elemente in den Grundzügen mit derjenigen von GRAFCET übereinstimmt. Die Erweiterbarkeit der PNML durch typspezifische Definitionen macht eine Überführung des formalen SIPN-Modells für GRAFCET grundsätzlich möglich, resultiert aber in einem nicht normkonformen PNML-Metamodell. Eine Überführung in normkonforme PNML-Metamodelle scheint aus momentaner Sicht

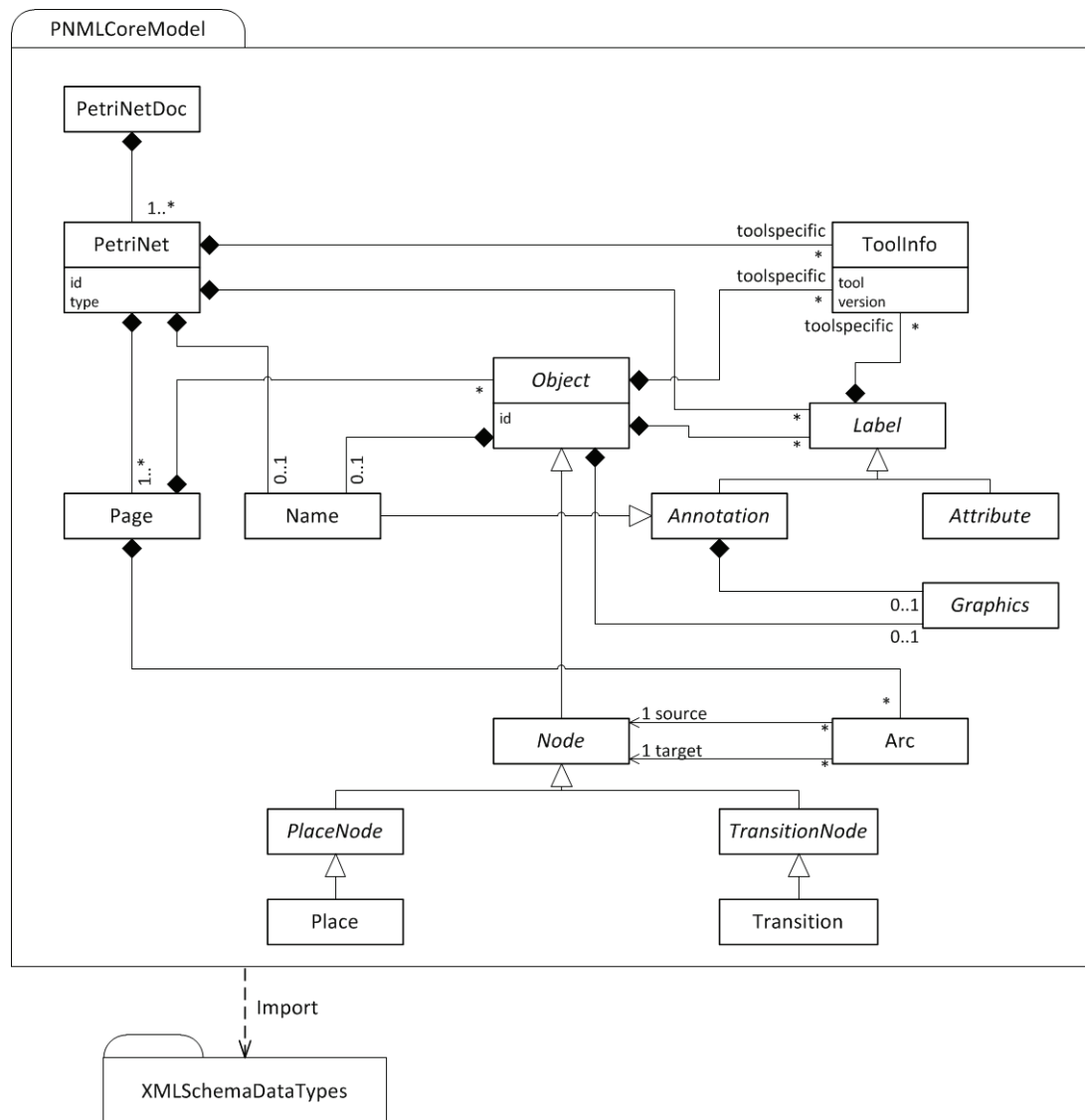


Abbildung 7-4: PNML Core Model in Anlehnung an ISO/IEC 15909-2

zielführender, da somit eine Zugriffsmöglichkeit über bestehende Petrinetz-Werkzeuge bestehen bleibt. Daraus lässt sich ableiten, dass die PNML gut als implementierungsunabhängige Notation geeignet ist.

7.2 Konzept für eine implementierungsunabhängige Notation in PNML

Gemäß den vorangegangenen Erläuterungen zu den einzelnen potentiellen Datenaustauschformaten für GRAFCET lassen sich die Ergebnisse, wie in Tabelle 7-2 dargestellt, zusammenfassen. So sind die Datenaustauschformate PCD und SCL nur in sehr konkreten Anwendungsfällen geeignet. Das fehlende notwendige Abstraktionsniveau verhindert eine Eignung als implementierungsunabhängige Notation für GRAFCET. *BatchML* bietet zwar die Möglichkeit, Schritte, Transitionen und Wirkverbindungen im Rahmen von Rezeptabläufen abzubilden. Weiterführende GRAFCET-Elemente können aber nicht berücksichtigt werden. Ähnliche Resultate ergeben sich für die Datenaustauschformate *PLCopenXML* und *AutomationML*. So lassen sich die Elemente eines *Basic-Grafcet*, aufgrund der Gemeinsamkeiten von GRAFCET und SFCs, Elementen in dem jeweiligen Format zuordnen. Die hierarchische Verknüpfung von Teil-Grafcets, beispielsweise in Form *zwangssteuernder Befehle* oder *einschließender Schritte*, lässt sich allerdings nicht abbilden.

Tabelle 7-2: Gegenüberstellung ausgewählter Datenaustauschformate bezüglich einer implementierungsunabhängigen Notation für GRAFCET

Eigenschaften des XML-basierten Datenaustauschformats	Datenaustauschformate					
	PCD	SCL	<i>BatchML</i>	<i>PLCopenXML</i>	<i>AutomationML</i>	PNML
frei verfügbar	○	+	+	+	+	+
Werkzeugunterstützung	○	○	+	○	+	○
GRAFCET-ähnliche Namensgebung für XML-Elemente	-	-	-	+	+	○
Abbildung von Basic-Grafcets	-	-	○	+	+	+
Abbildung hierarchisch strukturierter Grafcets	-	-	-	-	-	+
Besondere Kennzeichnung werkzeugspezifischer Daten	-	-	-	○	○	+
Abbildung grafischer Daten für GRAFCET-Elemente	-	-	-	+	+	+

Legende: +: erfüllt/vorhanden/möglich
 ○: teilweise erfüllt/teilweise vorhanden/teilweise möglich
 -: nicht erfüllt/nicht vorhanden/nicht möglich

Aufgrund der formalen Definition von GRAFCET als SIPN (→ 6) und den Ergebnissen der Analyse aus Tabelle 7-2 baut das Konzept für eine entsprechende implementierungsunabhängige Notation somit auf der PNML gemäß ISO/IEC 15909-2 auf. Die Anforderungen

(→ 7.1) werden größtenteils abgedeckt, so dass sich im Vergleich mit den anderen Datenaustauschformaten der höchste Abdeckungsgrad für die PNML ergibt.

Die Integration der Spezifikationsdaten kann einerseits durch die GRAFCET-spezifische Definition eines PNML-Metamodells erfolgen, welche als Erweiterung des PNML Core Models entwickelt wird. Wie bereits erläutert, würde das daraus resultierende eigenständige XML-Schema von den Vorgaben der ISO/IEC 15909-2 abweichen und eine Adaption an die bereits bestehende PNML-Infrastruktur erschweren. Eine andere Möglichkeit ergibt sich aus den in der ISO/IEC 15909-1 definierten *High-level Petrinetzen*, deren PNML-Repräsentation in Teil 2 des Standards enthalten ist. *High-level Petrinetze* stellen ein Dachformat der PNML dar. Die zugehörige XML-Syntax wird in Erweiterung des PNML Core Models in der Kernstruktur für *High-level Petrinetze*, der sogenannten *High-level Core Structure* [ISO/IEC 15909-2, S.15][#], definiert und ermöglicht die Integration verschiedener Petrinetz-Klassen durch typspezifische Einschränkungen. Die *High-level Core Structure* enthält eine Konkretisierung der im PNML Core Model lediglich abstrakt definierten Annotationen, die grundsätzlich jeder Stelle, Transition und Kante zugeordnet werden können. So werden jeder Stelle in einem Petrinetz ein nutzerspezifischer oder standardisierter Datentyp sowie eine Anfangsmarkierung zugewiesen, deren Wert mit dem Datentyp der Stelle vereinbar ist. Transitionen und Kanten werden durch Transitionsbedingungen und Kantengewichte ergänzt, die jeweils einer Funktion über einem definierten Wertebereich entsprechen. Nutzerspezifische Datentypen und weitere Deklarationen können für das Petrinetz oder das zugehörige Zeichenblatt hinterlegt werden.

Unter Berücksichtigung des formalen SIPN-Modells für GRAFCET bietet die *High-level Core Structure* eine konzeptionelle Grundlage für die Integration von GRAFCET in die PNML. Die dafür notwendige typspezifische Anpassung der *High-level Core Structure* ist in dem UML-Klassendiagramm in Abbildung 7-5 veranschaulicht. Die Definitionen der ISO/IEC 15909-2 zur Zusammensetzung syntaktisch korrekter Terme (*Terms*), Boolescher Ausdrücke (*Booleans*) und numerischer Operatoren (*Integers*) werden in das GRAFCET-spezifische PNML-Modell übernommen und tragen in wesentlichen Teilen zur Normkonformität bei. Weiterhin werden die für XML-Schema vordefinierten Datentypen [W3C XSD 1.1 PART 2][#] in das PNML-Modell integriert. Im GRAFCET-spezifischen PNML-Modell sind Annotationen nur für Stellen und Transitionen vorgesehen, welche die GRAFCET-Schritte und -Transitionen darstellen. Aufgrund des aktiven oder inaktiven Zustands eines GRAFCET-Schrittes wird jeder Stelle durch *Type* der Datentyp *Bool* zugewiesen. Ob es sich bei dem GRAFCET-Schritt um einen Initialschritt handelt, wird durch seine Anfangsmarkierung in der zu *HLMarking* zugehörigen Struktur hinterlegt. Jeder Transition wird eine Transitionsbedingung durch die mit der Annotation *Condition* assoziierten Struktur zugeordnet. Diese ist mit *toolspecific* gekennzeichnet, da die importierten vordefinierten Module für *Terms*, *Booleans* und *Integers* keine syntaktischen Definitionen für die steigende bzw. fallende Flanke einer Booleschen Variablen oder eines logischen Ausdrucks vorsehen, wie sie in GRAFCET möglich sind. Zeitabhängige Transitionsbedingungen im Sinne der IEC 60848 werden von den Definitionen ebenfalls nicht berücksichtigt. Durch die gemäß ISO/IEC 15909-2 zur Verfügung stehenden logischen Operatoren in *Terms* und *Booleans* wird lediglich die Boolesche Algebra in ihrer elementaren Form erfasst. Die erweiterten algebraischen Beziehungen für GRAFCET, wie beispielsweise in [LESAGE ET AL. 1996]

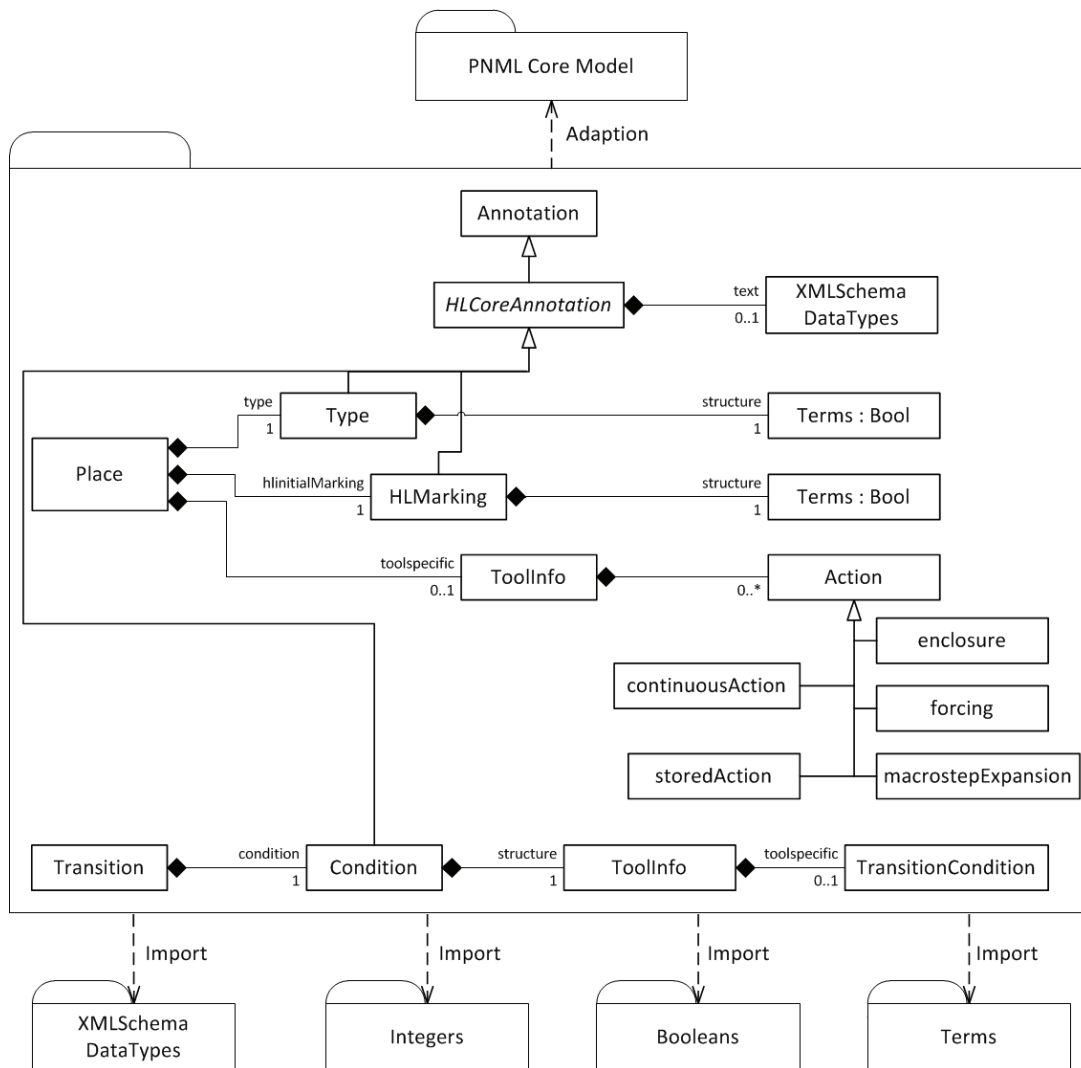


Abbildung 7-5: GRAFCET-spezifische Anpassung der High-level Core Structure, in Anlehnung an ISO/IEC 15909-2

beschrieben, können dadurch in PNML nicht vollständig abgedeckt werden. Um dennoch diese GRAFCET-spezifischen Eigenschaften abbilden zu können, werden Transitionsbedingungen im Rahmen dieses Konzepts als werkzeugspezifische Angaben deklariert. Für diese bestehen in der PNML keine syntaktischen Einschränkungen, so dass eine Abbildung aus GRAFCET vergleichsweise einfach möglich ist. Wirkverbindungen in GRAFCET werden generell als Kanten in PNML gespeichert, die einen Schritt oder eine Transition als Ursprung besitzen und zu einer Transition oder einem Schritt verlaufen. Besonderes Augenmerk muss im Rahmen dieses Konzeptes auf alternative und parallele Verzweigungen gelegt werden, die in GRAFCET durch einen teilweise gemeinsamen Fluss (alternative Verzweigung) und einen Doppelbalken (parallele Verzweigung) dargestellt werden. Da für *High-level Petrinetze* keine gesonderte Symbolik für alternative und parallele Verzweigungen definiert ist, müssen die entsprechenden Daten in besonderer Weise in PNML abgebildet werden. Im Rahmen der Transformation werden sowohl alternative als auch parallele Verzweigungen in ihre einzelnen Verbindungen aufgelöst und jeweils einer Kante in PNML zugeordnet (→ Abbildung 7-6). Die hauptsächlich der grafisch korrekten Darstellung dienende Information über einen teilweise gemeinsamen Fluss der Wirkverbindungen oder einen Doppelbalken wird in den werkzeugspezifischen Angaben der zugehörigen PNML-Seite abgelegt.

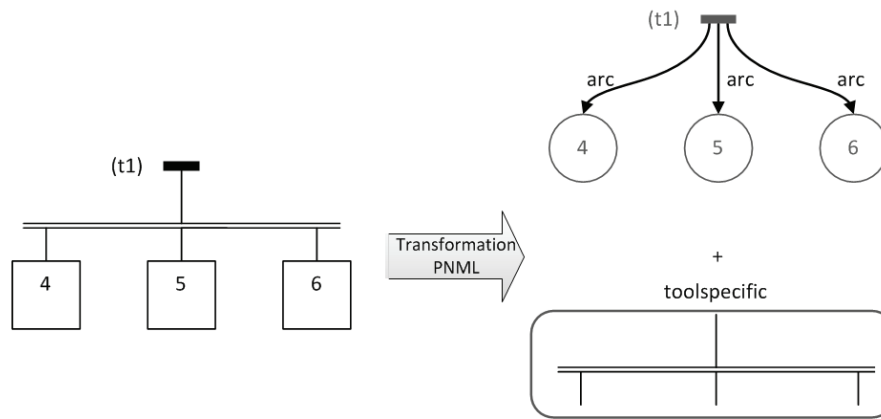


Abbildung 7-6: Abbildung paralleler GRAFCET-Verzweigungen in PNML

Mit den bisherigen Definitionen können alle Elemente eines Basic-Grafcet, ausgenommen Aktionen, in der PNML-Struktur eindeutig abgebildet werden. Aktionen werden in der momentanen Definition der *High-level Core Structure* nicht berücksichtigt. Wie für Transitionsbedingungen, ist aus diesem Grund auch in Bezug auf Aktionen eine pragmatische Lösung zur Abbildung in PNML zielführend. So wird in Bezug auf Stellen die werkzeugspezifische Angabe (*ToolInfo*) dazu genutzt, um die für kontinuierlich und speichernd wirkende Aktionen relevanten Informationen in PNML zu überführen. Das für Transitionsbedingungen (*TransitionCondition*), *kontinuierlich wirkende (continuousAction)* und *gespeichert wirkende Aktionen (storedAction)* genutzte Hilfsmittel, GRAFCET-spezifische Informationen in Form werkzeugspezifischer Daten in PNML abzubilden, wird im Rahmen des in dieser Arbeit vorgestellten Konzepts auch auf weiterführende hierarchische Konstrukte angewendet. So werden Makroschritte als Stellen abgebildet, in deren werkzeugspezifischen Angaben auf die zugehörige Makroschritt-Feinstruktur verwiesen wird (*macrostepExpansion*), sofern diese spezifiziert wurde. Die Makroschritt-Feinstruktur selbst wird als separates Petrinetz auf einem gesonderten Zeichenblatt in PNML aufgeführt. In gleicher Weise werden *einschließende Schritte* und die entsprechenden Teil-Grafcets der eingeschlossenen Schritte (*enclosure*) sowie *zwangssteuernde Befehle* und die zugehörigen zwangsgesteuerten Teil-Grafcets (*forcing*) in PNML überführt. Somit werden alle relevanten Elemente von GRAFCET in dem Konzept für eine implementierungsunabhängige Notation erfasst. Die PNML bietet hierzu eine geeignete Plattform, um als zentrale Datendrehscheibe eingesetzt zu werden. Aus den konzeptionellen Festlegungen dieses Kapitels müssen in einem nächsten Schritt Transformationsregeln für eine eindeutige Abbildung der GRAFCET-Elemente in die XML-Struktur der PNML abgeleitet werden. Sie sind Inhalt des nachfolgenden Abschnitts 7.3.

7.3 Umsetzung in die PNML-Notation nach ISO/IEC 15909-2

Die syntaktischen Definitionen zur XML-Repräsentation der PNML werden in ISO/IEC 15909-2 in der *Regular Language Description for XML New Generation (Relax NG)* dokumentiert. Relax NG ist eine XML-Schemasprache gemäß [ISO/IEC 19757-2][#] und kann zur Definition struktureller Vorgaben für XML-Dokumente verwendet werden. Eine Überführung von Relax NG in XML-Schema ist möglich und wird durch einige XML-Werkzeuge unterstützt. Im Folgenden wird allerdings nur auf das zugehörige XML-Schema der PNML eingegangen. Abbildung 7-7 zeigt die zugehörige Baumstruktur und die gemäß dem

Transformationskonzept vorgegebenen Zuordnungen der GRAFCET-Daten. Demnach besteht ein gültiges PNML-Dokument aus dem XML-Wurzelement *pnml*, welchem eine unbestimmte Menge an *net*-Elementen zugeordnet wird. Jedes *net*-Element repräsentiert eine GRAFCET-Spezifikation und besteht aus einem Namen (*name*), einem nutzerspezifischen Deklarationsteil für Variablen und Datentypen (*declaration*) und einer unbestimmten Anzahl von *page*-Elementen. Die *page*-Elemente wiederum repräsentieren einen Teil-Grafcet, eine Makroschritt-Feinstruktur oder einen Teil-Grafcet der eingeschlossenen Schritte. Sollte die Spezifikation nur einen Teil-Grafcet beinhalten, wie beispielsweise einen Basic-Grafcet oder normalisierten Grafcet, so besteht das PNML-Dokument aus lediglich einem *page*-Element. Jedes *page*-Element ist durch einen Namen gekennzeichnet und besteht aus einer unbestimmten Anzahl der XML-Elemente *place*, *transition* und *arc*, welche zur Abbildung der GRAFCET-Elemente Schritt, Transition und Wirkverbindung genutzt werden, sowie die Elemente *declaration* und *toolspecific* für zeichenblattspezifische Variablen- und Datentypdeklarationen und werkzeugspezifische Angaben. Die Daten bezüglich eines GRAFCET-Schrittes werden im *place*-Element unter Berücksichtigung der konzeptionellen Vorgaben aus Abbildung 7-5 abgelegt.

Am Beispiel eines GRAFCET-Schrittes mit *kontinuierlich wirkender Aktion* zeigt Abbildung 7–8, wie die GRAFCET-spezifischen Daten im Rahmen dieser Arbeit in der XML-Struktur der PNML abgebildet werden sollen. Die einzelnen XML-Elemente sind konform zu den Definitionen in ISO/IEC 15909-2. Die Angabe *bool* unter dem Element *type* beschreibt die Eigenschaft des GRAFCET-Schritts, entweder aktiv oder inaktiv zu sein. Bezogen auf die PNML bedeutet dies, dass die Stelle entweder eine Marke besitzt oder nicht.

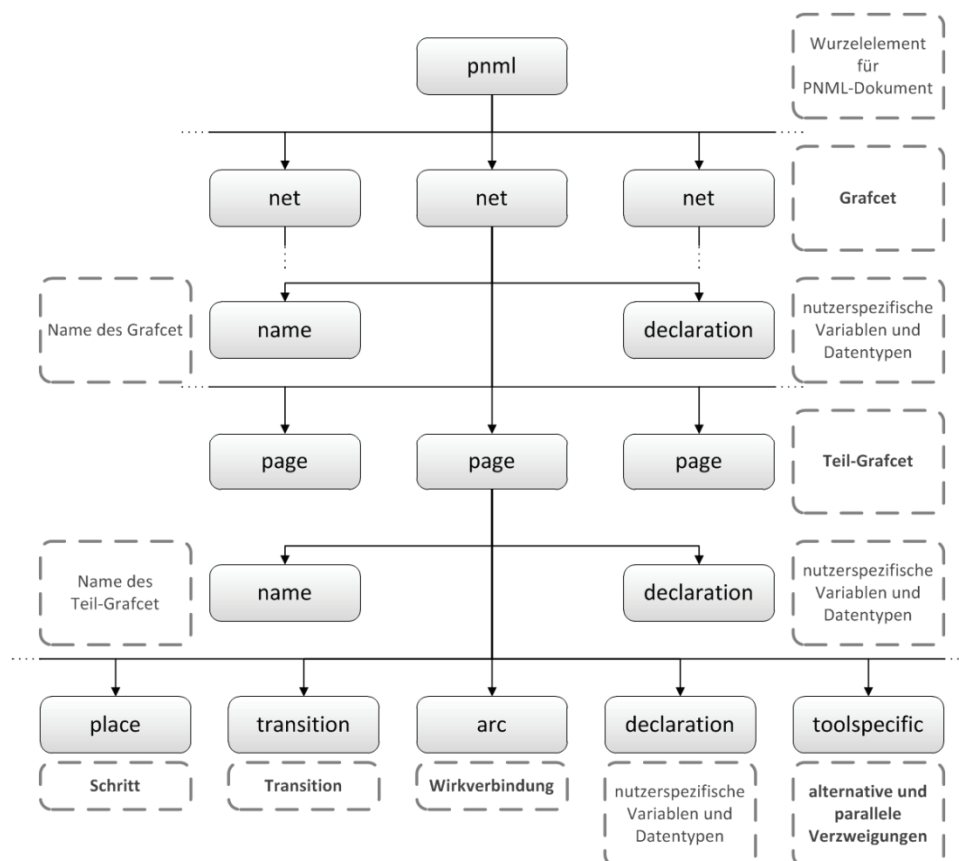


Abbildung 7-7: Strukturbau der PNML mit Zuordnung der GRAFCET-Elemente

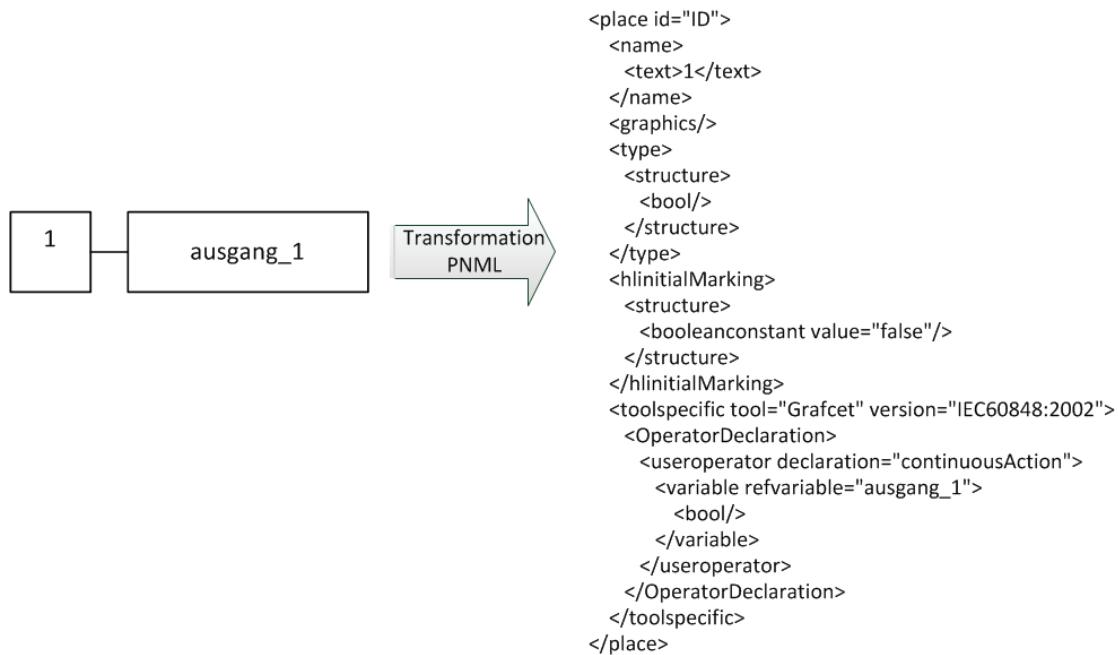


Abbildung 7-8: XML-Syntax gemäß PNML für einen GRAFCET-Schritt mit kontinuierlich wirkender Aktion

Jedes *place*-Element ist mit einer eindeutigen Identifikationsnummer versehen, die in dem *id*-Attribut hinterlegt wird. Da für alle XML-Elemente in PNML jeweils eine Identifikationsnummer vergeben wird, ist ein eindeutiger Bezug zu dem entsprechenden GRAFCET-Element vorhanden. Neben dem Namen des GRAFCET-Schrittes (*name*) und werkzeugspezifischen Angaben (*toolspecific*), werden die Schritt-Eigenschaften durch den Booleschen Datentyp in *type* und die Anfangsmarkierung in *hlnitialMarking* beschrieben. Zudem können optional grafische Informationen über die Position des GRAFCET-Schrittes im *place*-Element abgelegt werden. Hierbei ist allerdings zu beachten, dass diese Informationen werkzeugspezifisch sind, da die ISO/IEC 15909 keine Vorgaben bezüglich eines einheitlichen Koordinatensystems macht. Die Markierung der Stelle ist vom Datentyp *bool*. Die werkzeugspezifischen Angaben unter dem Element *toolspecific* enthalten alle Daten bezüglich des Wirkungsteils von GRAFCET, wie beispielsweise die *kontinuierlich wirkende Aktion* *ausgang_1* im Beispiel aus Abbildung 7-8.

Die nach diesem Prinzip definierten Regeln bilden die Grundlage für eine automatisierte Transformation von GRAFCET-Daten aus einem werkzeugspezifischen Datenformat in das offene, werkzeugunabhängige Datenformat der PNML (→ 9.2). Die PNML ist hinsichtlich GRAFCET eine geeignete implementierungsunabhängige Notation. Zwar können nicht alle GRAFCET-Elemente direkt in entsprechende XML-Elemente der PNML überführt werden. Eine Überbrückung dieser strukturellen Eigenschaft ist allerdings möglich, indem diese GRAFCET-spezifischen Informationen als werkzeugspezifische Informationen deklariert werden. Somit steht das resultierende PNML-Dokument sowohl GRAFCET- als auch Petrinetz-Werkzeugen offen, sofern Letztere die *toolspecific*-Elemente bei einem Import ignorieren. GRAFCET-Werkzeuge müssen diese Elemente durch die umgekehrte Anwendung der Abbildungsregeln in ihr werkzeugspezifisches Datenformat importieren. Aufgrund der bidirektionalen Anwendbarkeit der Abbildungsregeln ist dies grundsätzlich möglich.

8 Transformation von GRAFCET in Steuerungscode gemäß IEC 61131-3

8.1 Anforderungen an die Transformation

Zur Unterstützung einer automatischen Transformation des formalen SIPN-Modells für GRAFCET in ausführbaren Steuerungscode müssen entsprechende Abbildungsregeln definiert und deren Einschränkungen bzw. Anforderungen herausgestellt werden. Unter einer Transformation wird nachfolgend die Überführung einer Spezifikationsprache in eine Implementierungssprache verstanden, die durch die Anwendung von Transformationsregeln erfolgt. Konkret handelt es sich im Rahmen dieser Arbeit bei der Spezifikationsprache um GRAFCET gemäß IEC 60848 und bei der Implementierungssprache um ein Steuerungsprogramm, welches gemäß IEC 61131-3 erstellt wurde und in Form von Steuerungscode vorliegt. Die zugehörigen grundlegenden Transformationsregeln resultieren aus den Gemeinsamkeiten und spezifischen Eigenschaften der in [DIN EN 60848][#] und [DIN EN 61131-3][#] definierten Modelle. Dabei sollen die funktionalen Anforderungen eines durch GRAFCET spezifizierten Steuerungsablaufs im Vordergrund stehen.

Diese Transformationsregeln auf Modellebene sind zwar technologieunabhängig, für eine werkzeugunterstützte Umsetzung (→ 9.2) aber zu allgemein gefasst. Aus diesem Grund sind weitere technologieabhängige Regeln erforderlich, die eine Transformation auf Metamodellebene ermöglichen und auf einem möglichst offenen, XML-basierten Datenaustauschformat beruhen (→ 7.2). Als zugehörige Metamodelle für GRAFCET und IEC 61131-3 werden die XML-Schemata PNML (→ 7.1.2) und *PLCopenXML* verwendet. Ein entsprechendes Transformationskonzept muss die unterteilte Sichtweise der Modell- und Metamodellebene berücksichtigen (→ Abbildung 8–1), wobei die XML-Schemata PNML und *PLCopenXML* als technologische Randbedingungen festgelegt sind. Zwar wird im Rahmen dieser Arbeit ausschließlich die Transformation einer Spezifikation in eine Implementierung betrachtet, die Transformationsregeln sollten aber nach Möglichkeit bidirektional ausgelegt sein und somit auch die Voraussetzungen für die Transformation einer Implementierung in eine Spezifikation schaffen. Als Implementierung sind aufgrund der Gemeinsamkeiten mit GRAFCET insbesondere SFCs vorgesehen, so dass aus äquivalentem dynamischem Verhalten von GRAFCET- und SFC-Konstrukten einfache, direkte Abbildungsvorschriften resultieren und gleichzeitig die grafische Eingängigkeit erhalten bleibt.

Weitere Anforderungen an die Transformation eines Grafcet in IEC 61131-3 konformen Steuerungscode ergeben sich aus modellspezifischen Annahmen hinsichtlich GRAFCET und SFC. Sie beruhen einerseits auf bereits diskutierten Konstruktionsregeln für Grafkets (→ 6.1.1) und andererseits auf dem vorhandenen Interpretationsspielraum der IEC 61131-3 bezüglich SFCs. Für die Entwicklung eindeutiger Transformationsregeln dürfen nur solche Konstrukte der IEC 61131-3 verwendet werden, deren dynamisches Verhalten eindeutig und somit implementierungsunabhängig definiert ist. Die entsprechenden Einschränkungen müssen folglich auch hinsichtlich GRAFCET geprüft und umgesetzt werden. So muss es sich bei dem zu transformierenden Grafcet um einen gemäß IEC 60848 korrekten und wohlgeformten Grafcet (*sound Grafcet*) handeln.

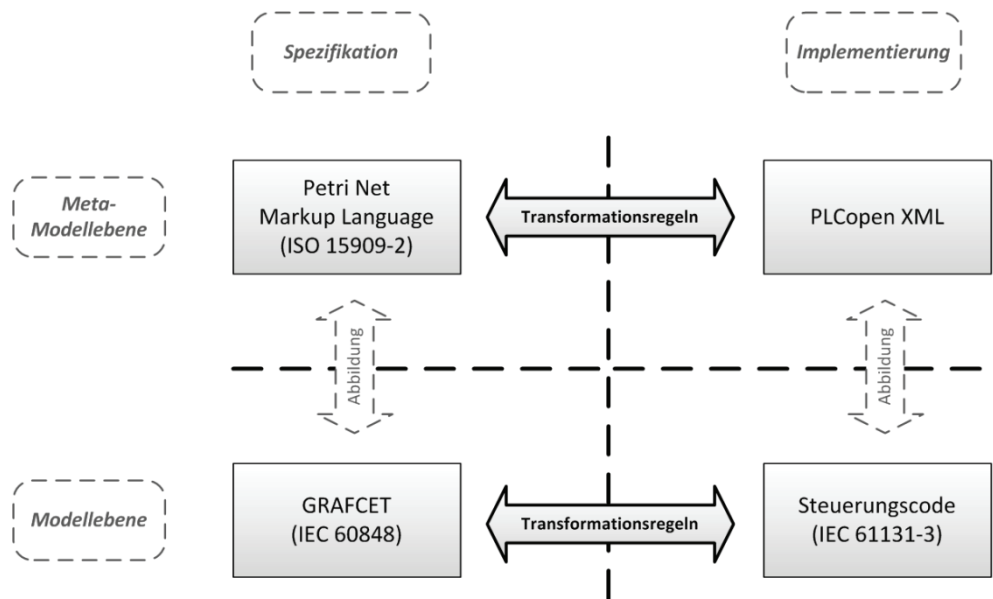


Abbildung 8-1: Ebenensicht für das Transformationskonzept

Dies beinhaltet auch, dass die in IEC 60848 festgelegten Kriterien und Hinweise zur Erstellung eines Grafcet beachtet werden, wie beispielsweise die „[...] Festlegung einer Zuordnung für eine Ausgangsvariable (gespeichert wirkende Art) schließt diese Ausgangsvariable von jeder Zuweisung aus (kontinuierlich wirkende Art).“ [DIN EN 60848, S.16][#]. Um Konflikte der Ausgangswertigkeit von Variablen zu vermeiden, sollten Schritte, die gleichzeitig aktiv sein können, nicht dieselben Ausgangsvariablen manipulieren. Im Falle *kontinuierlich wirkender Aktionen* können definitionsgemäß keine solchen Konflikte auftreten, wohl aber bei *gespeichert wirkenden Aktionen*.

Schrittvariablen in Transitionsbedingungen und Zuweisungsbedingungen sollten in einem Basic-Grafcet und besonders in einem hierarchisch strukturierten Grafcet mit Bedacht gewählt und als lokale Variablen verwendet werden. Die Einschränkung der lokalen Verwendung bezieht sich auf den jeweils zugehörigen Teil-Grafcet des Schrittes. Somit werden über die Einschließung (→ 6.2.2) und Zwangssteuerung (→ 6.2.3) hinausgehende Verknüpfungen zwischen Teil-Grafcets vermieden. Sie werden ausschließlich durch *einschließende Schritte* und *zwangssteuernde Befehle* spezifiziert. Weiterhin sollten die aus den Überlegungen zu einem transparenten Steuerungsentwurf mit SFCs [FREY & LITZ 1997] resultierenden Einschränkungen, wie beispielsweise das Verbot von Sprüngen aus und zwischen parallel verlaufenden Schrittkettensequenzen, auch in Bezug auf einen Grafcet Berücksichtigung finden. Somit kann bereits bei der Erstellung einer Spezifikation unerwartetes dynamisches Verhalten im Steuerungsablauf konstruktiv vermieden werden.

Eine wesentliche Anforderung an die Transformation eines Grafcet in eine IEC 61131-3 konforme Implementierung im Rahmen dieser Arbeit ist die auf den Gemeinsamkeiten von GRAFCET und SFC basierende Entwicklung geeigneter Transformationsregeln. Wie bereits erläutert (→ 3.3), ergeben sich allerdings hersteller- bzw. werkzeugabhängig unterschiedliche Interpretationen für das Ausführungsmodell von SFCs [HELLGREN ET AL. 2005], wie beispielsweise bei der Reihenfolge der Auswertung von Transitionsbedingungen in alternativen Verzweigungen. Die Transformationsregeln müssen dennoch eindeutig sowie hersteller- und werkzeugunabhängig sein, so dass der resultierende Steuerungscode

grundsätzlich in jede SPS importiert werden kann und dort dasselbe dynamische Verhalten hervorruft. Eine weitere wesentliche Anforderung an die Transformation ist somit, dass die in Betracht gezogenen SFCs die Eigenschaften eines in [BAUER 2003] definierten *implementierungsunabhängigen SFC* erfüllen.

Neben den Gemeinsamkeiten von GRAFCET und SFC müssen auch für die grundsätzlichen Unterschiede beider Modelle geeignete Transformationsregeln entwickelt werden, so dass beispielsweise Grafquets mit mehreren Initialschritten oder hierarchisch strukturierte Grafquets in SFC-basierten Steuerungscode abgebildet werden können.

8.2 Konzept für die Transformation

8.2.1 Transformation auf Modellebene

Im Rahmen der Transformation existieren zahlreiche Möglichkeiten für die Überführung von Informationen bezüglich des gewünschten Steuerungsablaufs, die in einem Grafquet spezifiziert sind, in Steuerungscode, welcher die Anforderungen der IEC 61131-3 erfüllt. So stehen gemäß dem Programmiermodell der IEC 61131-3 [DIN EN 61131-3, S.20][#] mit AWL, ST, KOP, FBS und SFC insgesamt fünf Programmiersprachen zur Verfügung, die zur Deklaration von Funktionen, Funktionsbausteinen und Programmen verwendet und miteinander kombiniert werden können. Aus einem Programm heraus können sowohl standardisierte als auch nutzerspezifische Funktionen und Funktionsbausteine aufgerufen werden. Konfigurationen können aus mehreren POEs bestehen und stellen auf einer konkreten Hardware ausführbare Steuerungsprogramme dar. Die Angabe notwendiger technologiespezifischer Hardware-Eigenschaften erfolgt hierbei durch sogenannte Konfigurationselemente.

In einem ersten Schritt muss hinsichtlich des Konzepts für die Transformation auf Modellebene festgelegt werden, welche GRAFCET-Informationen wo in einer IEC 61131-3-Konfiguration abgelegt werden sollen. Die Gemeinsamkeiten von GRAFCET und SFC sollen im Rahmen des Konzepts explizit ausgenutzt werden. Abbildung 8-2 zeigt eine schematische Gegenüberstellung des Informationsgehalts eines Grafquets und des Programmiermodells gemäß IEC 61131-3. Die Verbindungen zwischen den einzelnen Strukturelementen zeigen auf, wie die GRAFCET-Informationen konzeptionell abgebildet werden sollen. Die grau markierten Konstrukte der Makroschritte, *einschließenden Schritte* und *zwangsteuernden Befehle* haben keine unmittelbare Entsprechung in der IEC 61131-3 und müssen durch geeignete Hilfskonstruktionen überführt werden. Das Konzept sieht vor, dass ein Grafquet in einen einzigen SFC überführt wird, der ein Programm im Sinne der gemäß IEC 61131-3 definierten POEs darstellt. Sollte der Grafquet Makroschritte und zugehörige Feinstrukturen enthalten oder aus mehreren Teil-Grafquets bestehen, die beispielsweise durch *zwangsteuernde Befehle* oder *einschließende Schritte* miteinander verknüpft sind, so muss zunächst eine Normalisierung erfolgen. Es bestünde beispielsweise die Möglichkeit, die einzelnen Teil-Grafquets jeweils in nutzerspezifischen Funktionsbausteinen zu kapseln. Allerdings müssten, zur Verknüpfung der einzelnen Funktionsbausteine und zur Gewährleistung eines äquivalenten Verhaltens, zusätzliche Variablen an den Funktionsbaustein-Schnittstellen definiert werden.

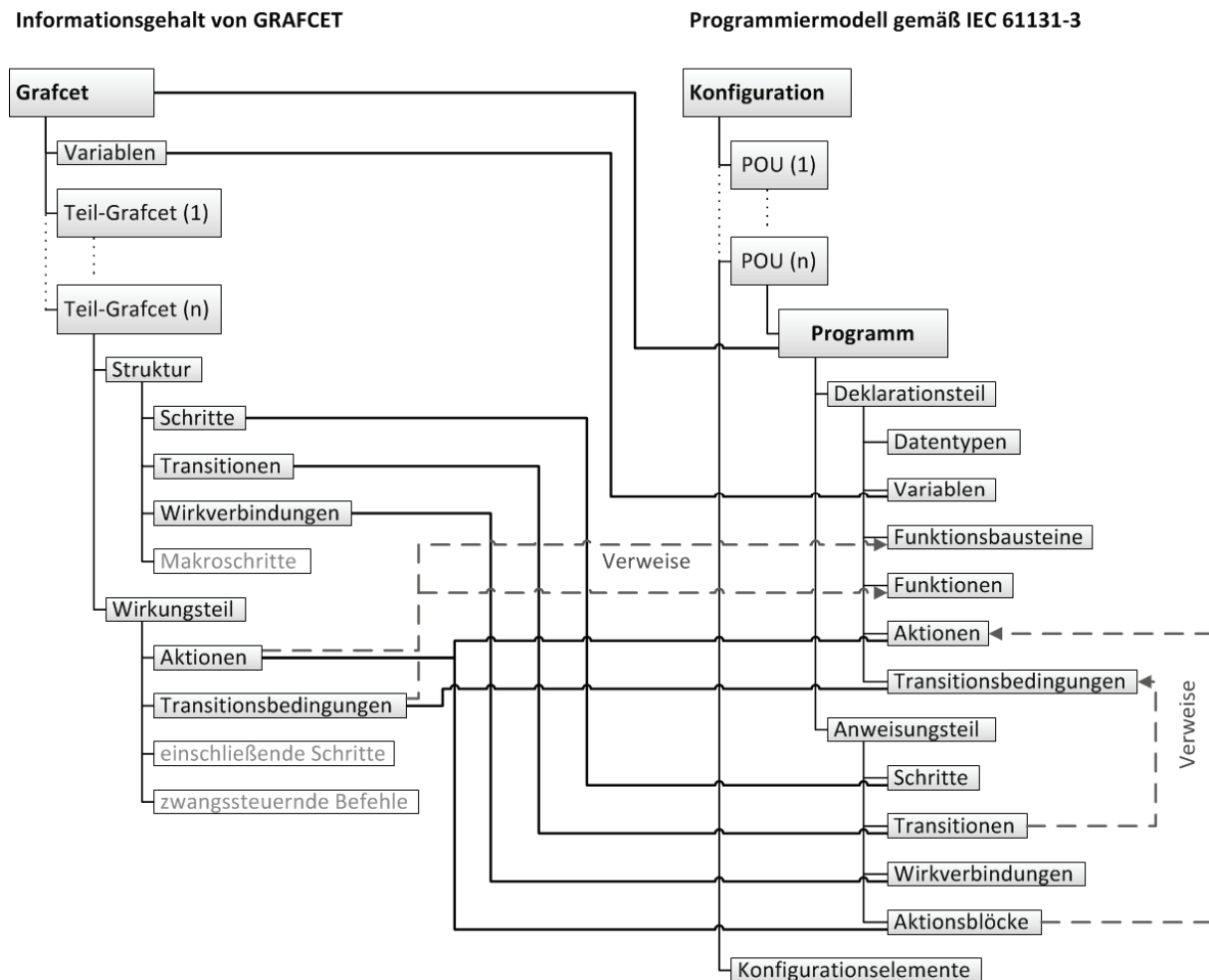


Abbildung 8-2: Überführung von GRAFCET in das Programmiermodell der IEC 61131-3

Dies würde zu Einschränkungen bezüglich der Transparenz des Steuerungscode führen, da die einzelnen Zusammenhänge zwischen den Funktionsbausteinen nicht offensichtlich durch Wirkverbindungen, sondern implizit durch Schnittstellenvariablen und erweiterte Transitionsbedingungen gegeben wären. Zudem würde eine übergeordnete Koordination der einzelnen Funktionsbausteine durch den Aufruf innerhalb eines Programms weiteren zusätzlichen Aufwand erfordern, um ein GRAFCET-äquivalentes dynamisches Verhalten sicherzustellen.

Die in einem Grafcet spezifizierten symbolischen Eingangs- und Ausgangsvariablen können im Deklarationsteil des Programms übernommen werden, müssten hinsichtlich einer ausführbaren Konfiguration aber noch durch hardware-spezifische Adressbezüge ergänzt werden. Weitere Angaben im Deklarationsteil des Programms beziehen sich auf Datentypen, Funktionen und Funktionsbausteine, wobei es sich sowohl um standardisierte als auch um nutzerspezifische Angaben handeln kann. Aufgrund der Eigenschaften der logischen und numerischen Variablen eines Grafcet ist die Deklaration nutzerspezifischer Datentypen im Rahmen der Transformation nicht notwendig. Zur Definition geeigneter Hilfskonstruktionen für nicht direkt transformierbare GRAFCET-Konstrukte können standardisierte Funktionen und Funktionsbausteine instanziiert werden, deren Verhalten eindeutig in IEC 61131-3 definiert ist. Eine wesentliche Herausforderung bei der Entwicklung geeigneter Transformationsregeln besteht demnach darin, solche Hilfskonstruktionen in der Weise zu definieren, dass gemäß den Definitionen der IEC 60848 und IEC 61131-3 äquivalentes Verhalten gewährleistet ist.

Die Deklarationen von GRAFCET-Aktionen, die keine direkte Entsprechung als SFC-Aktion aufweisen, sowie Transitionsbedingungen können ebenfalls im Deklarationsteil des Programms abgelegt und in einer der fünf Programmiersprachen implementiert werden. Aus der SFC-Struktur heraus wird in diesem Fall über den Aktionsblock und einen passenden Aktionsqualifier oder über den Namen der Transitionsbedingung auf die entsprechende Deklaration verwiesen. Aufgrund der funktionalen Sichtweise im Zuge der Transformation stellen die textbasierten Programmiersprachen, insbesondere ST, hierzu kompakte Sprachkonstrukte zur Verfügung. Sollte eine GRAFCET-Aktion direkt in eine SFC-Aktion überführt werden können, das durch die GRAFCET-Aktion spezifizierte Verhalten also äquivalent durch einen Aktionsqualifier beschrieben sein, so wäre eine Deklaration der Aktion nicht erforderlich. Es würde ausreichen, über den Aktionsblock und unter Angabe des passenden Aktionsqualifiers auf die entsprechende Ausgangsvariable zu verweisen.

In einem zweiten Schritt muss die bisher noch nicht berücksichtigte Eigenschaft eines Grafcet, mehr als einen Initialschritt enthalten zu können, in das bisher vorgestellte Konzept integriert werden, da in einem SFC per Definition lediglich ein Initialschritt existieren darf. Gemäß [BAUER 2003] sollte dieser SFC-Initialschritt keine assoziierten Aktionen aufweisen, um implementierungsunabhängig zu sein. In einer weiteren Vorstufe zur eigentlichen Transformation wird der im ersten Schritt strukturell normalisierte Grafcet nun im zweiten Schritt in einen transformierbaren Grafcet überführt, der lediglich einen Initialschritt ohne assoziierte Aktionen besitzt. Diese zweistufige Umwandlung ist deshalb notwendig, weil GRAFCET und IEC 61131-3 grundsätzlich unterschiedliche Hierarchiekonzepte aufweisen.

Im Rahmen der Normalisierung der Initialschritte wird dem normalisierten Grafcet ein Schritt hinzugefügt, welcher keine eingehenden Wirkverbindungen besitzt und im Nachbereich mit einer einzigen Transition verbunden ist. Dem Schritt wird die Eigenschaft eines Initialschritts zugewiesen. Die zur Transition zugehörige Transitionsbedingung ist die immer erfüllte Transitionsbedingung (*true*). Der Nachbereich der Transition besteht aus der Menge der Initialschritte des Grafcet, die bei Übergang des normalisierten Grafcet in den transformierbaren Grafcet den Status eines Initialschrittes verlieren. Somit kann die Anforderung an einen SFC, nur einen Initialschritt zu besitzen, durch den transformierbaren Grafcet pragmatisch und mit geringem Aufwand erfüllt werden (→ Abbildung 8-3).

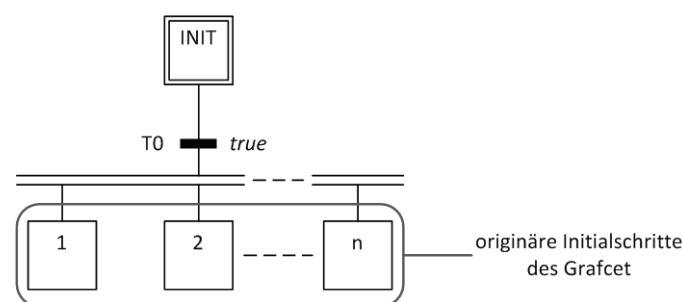


Abbildung 8-3: Normalisierung der Initialschritte im Zuge der Transformation

In einem dritten, abschließenden Schritt werden die aus den zuvor beschriebenen Verknüpfungen resultierenden Transformationsregeln auf den transformierbaren Grafcet angewendet, mit dem Ergebnis eines IEC 61131-3 konformen Steuerungsprogramms. Die somit beschriebenen wesentlichen Schritte des Transformationskonzepts auf Modellebene werden in Abbildung 8-4 veranschaulicht.

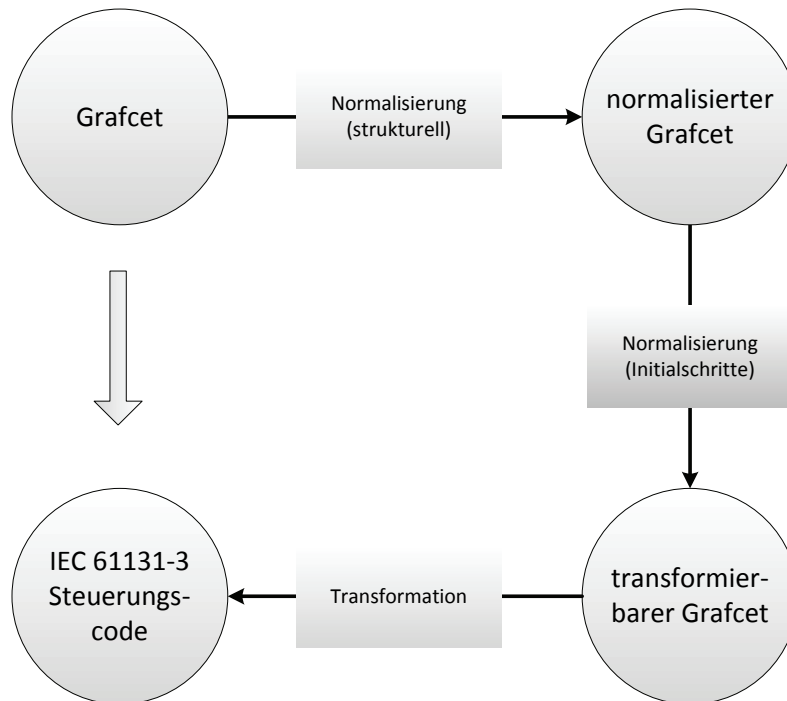


Abbildung 8-4: Transformationskonzept auf Modellebene

8.2.2 Transformation auf Metamodellebene

Für die Transformation auf Metamodellebene wurde die PNML gemäß ISO/IEC 15909-2 aufgrund des formalen SIPN-Modells für GRAFCET sowie den Ausführungen zu einer implementierungsunabhängigen Notation (→ 7) bereits als Ausgangs-Datenaustauschformat festgelegt. Auch das Ziel-Datenaustauschformat steht in Form des *PLCopenXML*-Schemas [PLCOPEN 2009][#] fest. Das im Rahmen dieses Kapitels beschriebene Konzept für die Transformation auf Metamodellebene bildet somit die Grundlage für die systematische Überführung eines als PNML-Datei vorliegenden Grafcet in eine *PLCopenXML*-Datei. Grundsätzlich müssen die konzeptionellen Rahmenbedingungen bezüglich der Transformation auf der Modellebene auch für die Transformation auf der Metamodellebene beachtet werden, denn nach wie vor wird der Inhalt eines Grafcet in das Programmiermodell der IEC 61131-3 übertragen. Die XML-basierten Datenaustauschformate liefern darüber hinausgehend eine Infrastruktur, mit deren Hilfe die Informationen bezüglich der Spezifikation in einem offenen, werkzeugunabhängigen Format gespeichert werden können, so dass die Daten durch rechnergestützte, formale Methoden bearbeitet und in ein entsprechend offenes, werkzeugunabhängiges Format für die Implementierung überführt werden. Die hierfür notwendigen Transformationsregeln beziehen sich dann nicht mehr auf die jeweiligen Elemente der Modellebene, sondern auf die sie repräsentierenden XML-Elemente der PNML und *PLCopenXML*.

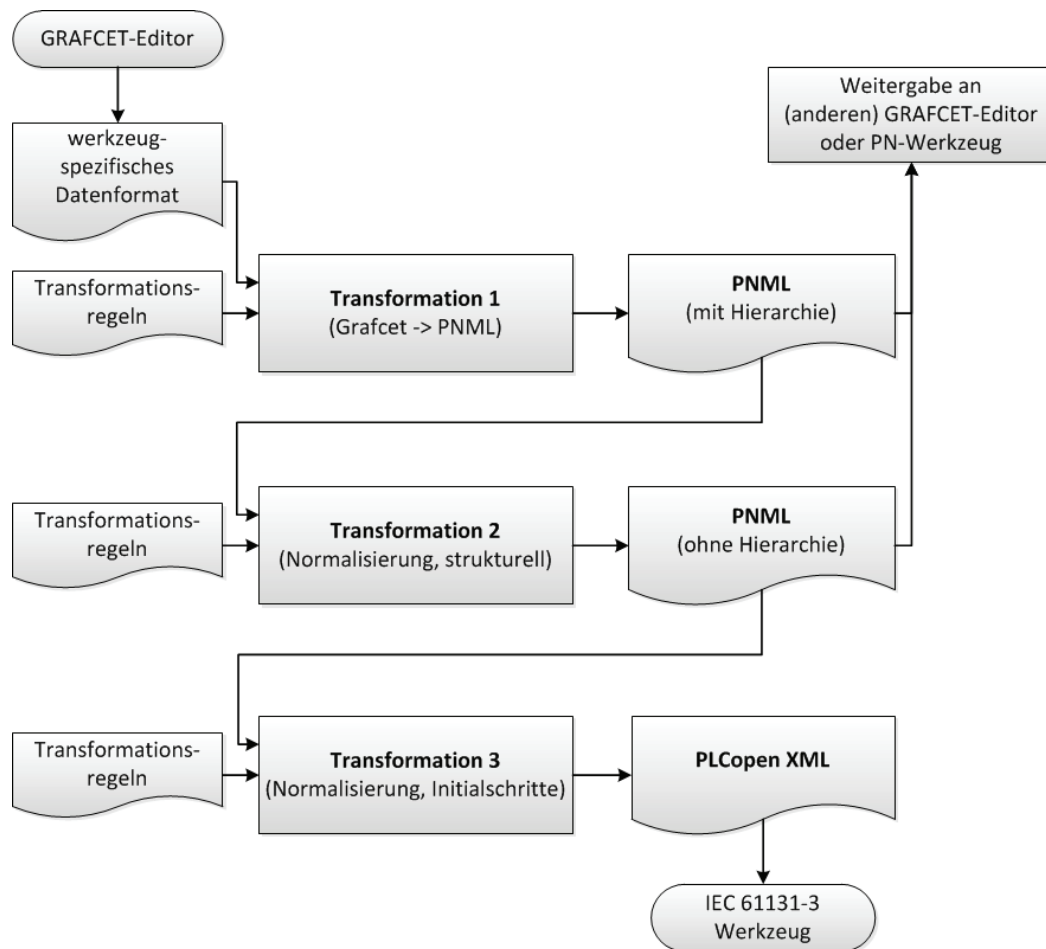
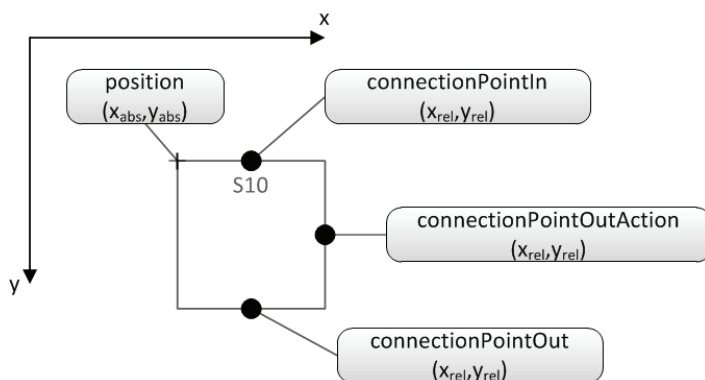


Abbildung 8-5: Transformationskonzept auf Metamodellebene

Zunächst muss in Vorbereitung der Transformation in das PLCopenXML-Format sichergestellt sein, dass die in einem werkzeugspezifischen Datenformat vorliegenden GRAFCET-Daten extrahiert und in einer PNML-Datei abgespeichert werden. Diese, in Abbildung 8-5 mit *Transformation 1* gekennzeichnete Transformation erfordert Wissen über die Struktur und inhaltliche Zusammensetzung des werkzeugspezifischen Datenformats, welches in den zugehörigen Transformationsregeln dokumentiert und angewendet wird. Das Ergebnis der Transformation 1 ist eine gemäß den Anforderungen der PNML gültige XML-Datei, die hierarchische Strukturen eines Grafcet explizit enthält. In einem weiteren Schritt erfolgt, in Anlehnung an die in Abbildung 8-4 dargestellte methodische Vorgehensweise, die

Abbildung 8-6: Grafische Informationen am Beispiel eines Schrittes, gemäß [PLCopen 2009][#]

Normalisierung in struktureller Hinsicht (*Transformation 2*), so dass die vorliegenden Informationen über Makroschritte, *einschließende Schritte* und *zwangsteuernde Befehle* (\rightarrow 6.2) durch die entsprechenden Konstrukte eines Basic-Grafcet (\rightarrow 6.1) ersetzt werden. Die zugehörigen Transformationsregeln werden auf die aus der Transfor-

mation 1 resultierende PNML-Datei angewendet. Hieraus entsteht ein PNML-Dokument, welches alle Informationen bezüglich des Grafcet in einem *page*-Element und somit keine hierarchische Struktur mehr enthält, im Sinne der PNML aber weiterhin gültig ist. Die Normalisierung der Initialschritte (*Transformation 3*) wird erst im Rahmen der Transformation der PNML-Datei in eine *PLCopenXML*-Datei gemäß den zugehörigen Transformationsregeln durchgeführt. Dies liegt darin begründet, dass die PNML die Initialsituation eines Grafcet durch eine entsprechende Anfangsmarkierung der Schritte, hinterlegt in den *step*-Elementen, abbilden kann und erst bei Übergang in das *PLCopenXML*-Format eine Normalisierung auf einen Initialschritt notwendig wird.

Die bisher beschriebenen Bestandteile des Konzepts zeigen eine dreistufige systematische Vorgehensweise für die Transformation eines als PNML-Datei vorliegenden Grafcets in eine *PLCopenXML*-Datei durch Adaption des Transformationskonzeptes auf Modellebene. Im Rahmen einer Transformation auf Metamodellebene müssen darüber hinaus spezifische Eigenschaften des *PLCopenXML*-Formats in das Konzept mit einbezogen werden, um davon ausgehend eindeutige Transformationsregeln abzuleiten. So beinhaltet das *PLCopenXML*-Format in Bezug auf SFCs, neben den funktionalen Angaben, Informationen darüber, wie und wo die grafischen SFC-Elemente auf einer Zeichenoberfläche dargestellt werden. Beispielsweise sind für einen SFC-Schritt, dem eine Aktion assoziiert wurde und der im Vor- und Nachbereich mit einer Transition verbunden ist, vier Positionsangaben notwendig (→ Abbildung 8-6). Neben einer absoluten Positionsangabe sind weitere drei relative Positionsangaben für grafische Verbindungspunkte als notwendige Angaben im *PLCopenXML*-Schema vorgeschrieben. Zu beachten ist, dass es sich jeweils um werkzeugspezifische zweidimensionale Koordinatenangaben handelt. Nur durch die erforderliche Dokumentation entsprechender Skalenfaktoren im Kopfteil der *PLCopenXML*-Datei können diese Koordinatenangaben in andere werkzeugspezifische Koordinatensysteme transformiert werden.

Aus der funktionalen Sichtweise einer Steuerungsspezifikation in GRAFCET, welche im Rahmen der Transformation in Steuerungscode überführt werden soll, stellt diese unverhältnismäßige Gewichtung der grafischen Informationen im *PLCopenXML*-Format ein wesentliches Hindernis für eine automatische Generierung dar. Ein wesentlicher Teil des Algorithmus in Transformation 3 müsste die aufwendigen grafischen Berechnungen durchführen, was die Berechnung des Verlaufes von Wirkverbindungen mit einschließt, und wäre somit werkzeugspezifisch. Aus diesem Grund verfolgt das im Rahmen dieser Arbeit vorgestellte Konzept für die Transformation auf Metamodellebene einen anderen Weg und fokussiert auf eine funktionale Beschreibung der SFC-Struktur. So wird die Struktur eines SFCs in Form von Schritten, Transitionen und Wirkverbindungen nicht in ihrer grafischen, sondern in ihrer textuellen Repräsentation als ST gemäß IEC 61131-3 beschrieben und in einer nach dem *PLCopenXML*-Schema gültigen Datei abgelegt. Die textuelle Repräsentation von Schritten, Transitionen und Wirkverbindungen speichert die funktionalen Zusammenhänge eines Grafcet im Zuge der Transformation als kompakten Steuerungscode. Dieser Aspekt ist ein weiterer bedeutungsvoller Bestandteil des hier vorgestellten Transformationskonzepts auf Metamodellebene.

8.3 Definition der Transformationsregeln

8.3.1 Transformationsregeln auf Modellebene

Aufgrund der vorab durchgeführten Schritte zur Normalisierung (→ Abbildung 8-4) müssen die Transformationsregeln, neben den grundlegenden Elementen der Schritte, Transitionen und Wirkverbindungen, ausschließlich Transitionsbedingungen sowie *kontinuierlich* und *gespeichert wirkende Aktionen* berücksichtigen. Der Definition solcher Transformationsregeln auf Modellebene geht die Analyse des dynamischen Verhaltens von GRAFCET- und SFC-Konstrukten gemäß der Standards IEC 60848 und IEC 61131-3 voraus. Unter einem GRAFCET-Konstrukt wird eine bestimmte Form einer Transitionsbedingung oder kontinuierlich oder speichernd wirkender Aktionen verstanden. Die Analyse zielt darauf ab, die Gemeinsamkeiten von GRAFCET und SFC bezüglich dieser Konstrukte im Rahmen der Transformation weitestmöglich zu berücksichtigen. Weisen beide Konstrukte äquivalentes dynamisches Verhalten auf, so besteht die zugehörige Transformationsregel aus einer direkten Entsprechung. Zusätzliche Hilfskonstruktionen zur Ergänzung des SFCs werden nur dann benötigt, wenn es für das jeweilige GRAFCET-Konstrukt keine direkte Entsprechung gibt. Insgesamt wurden 28 Transformationsregeln definiert (→ Anhang D). Aufgrund ihrer großen Anzahl werden nachfolgend einige Beispiele für Transformationsregeln bezüglich der Modellebene erläutert.

Zunächst soll das Augenmerk auf diejenigen GRAFCET-Konstrukte gelegt werden, für die eine direkte Entsprechung in SFC möglich ist. Neben den Schritten, Transitionen und Wirkverbindungen existieren teilweise in Bezug auf *kontinuierlich* und *gespeichert wirkende Aktionen* ebenfalls Transformationsregeln, die unmittelbar in SFC-Konstrukte überführt werden können. So besitzen einfache *kontinuierlich wirkende Aktionen*, ohne Zuweisungsbedingungen, und SFC-Aktionen mit dem Aktionsqualifier *N* äquivalentes dynamisches Verhalten (→ Abbildung 8-7). In beiden Fällen handelt es sich um *kontinuierlich wirkende Aktionen* zur Aktivierung einer Booleschen Ausgangsvariablen, deren Ausführungsdauer unmittelbar mit dem aktiven Zustand des ihnen assoziierten Schrittes verknüpft ist.

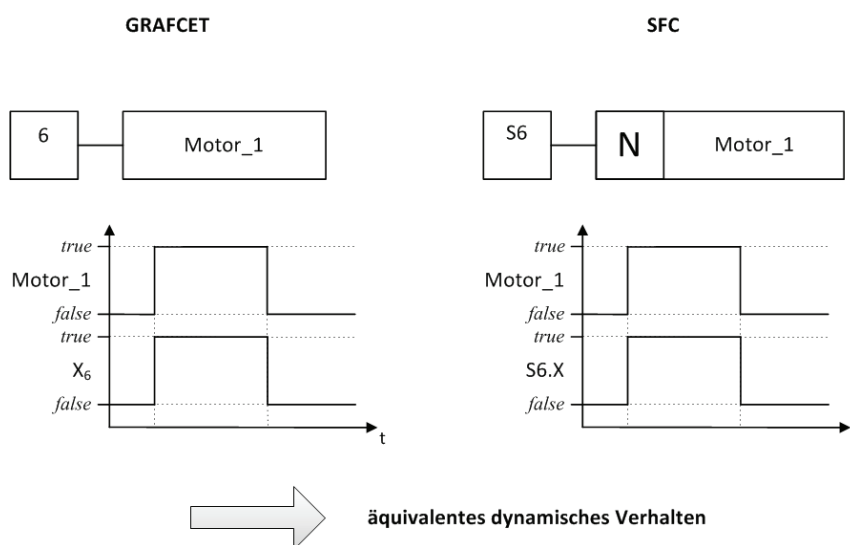


Abbildung 8-7: Äquivalentes dynamisches Verhalten kontinuierlich wirkender Aktionen

Auch *gespeichert wirkende Aktionen*, die an das Ereignis der Aktivierung des ihnen assoziierten Schrittes gekoppelt sind, und einer Booleschen Ausgangsvariablen den Wert *true* oder *false* zuordnen, entsprechen in ihrem dynamischen Verhalten einer SFC-Aktion mit dem Aktionsqualifier *S* (wenn *true*) oder *R* (wenn *false*). Der Aktionsname verweist diesbezüglich auf eine Boolesche Ausgangsvariable. Diese direkten Entsprechungen sind in den Transformationsregeln 21 und 22 enthalten (\rightarrow Anhang D). Weitere direkte Entsprechungen ergeben sich beispielsweise für zeitverzögerte und zeitbegrenzte *kontinuierlich wirkende Aktionen*, deren zeitabhängige Zuweisungsbedingung an die Schrittvariable des assoziierten Schrittes gebunden ist. Das in diesem Falle spezifizierte dynamische Verhalten entspricht SFC-Aktionen, die mit dem Aktionsqualifier *D* oder *L* verknüpft sind, siehe Transformationsregeln 14 und 15 (\rightarrow Anhang D). Darüber hinaus existieren zahlreiche GRAFCET-Konstrukte, deren gemäß IEC 60848 definiertes dynamisches Verhalten nicht direkt in SFC abgebildet werden kann. Ein äquivalentes dynamisches Verhalten kann nur dann sichergestellt werden, wenn geeignete Hilfskonstruktionen eingesetzt werden, die in dieser Arbeit in ST beschrieben werden. Wie bereits in Abbildung 8-2 aufgezeigt, handelt es sich bei diesen Hilfskonstruktionen um Deklarationen für Aktionen und Transitionsbedingungen.

Sämtliche Transitionsbedingungsarten von GRAFCET werden inhaltlich im dafür vorgesehenen Deklarationsteil des IEC 61131-3 Programmiermodells durch entsprechende ST-Anweisungen beschrieben. Die zugehörigen Transitionen werden im Anweisungsteil des Programms abgelegt und verweisen auf die im Deklarationsteil vorhandene Transitionsbedingung, wie am Beispiel von Transformationsregel 7 (\rightarrow Anhang D) in Abbildung 8–8 verdeutlicht wird. Sollten Ereignisse, wie beispielsweise die steigende oder fallende Flanke einer Eingangsvariablen, in einer Transitionsbedingung spezifiziert worden sein, so wird dieses Verhalten im Programmiermodell der IEC 61131-3 durch Instanzen der standardisierten Funktionsbausteine *R_TRIG* (für \uparrow) und *F_TRIG* (für \downarrow) eingesetzt. Bezüglich zeitabhängiger Transitionsbedingungen werden die standardisierten Funktionsbausteine *TON* (Einschaltverzögerung) und *TOF* (Ausschaltverzögerung) verwendet. Alle Funktionsbausteine sind in IEC 61131-3 eindeutig definiert, wodurch das durch sie beschriebene dynamische Verhalten implementierungsunabhängig ist. Eine ähnliche Vorgehensweise ergibt sich für diejenigen *kontinuierlich* und *gespeichert wirkenden* GRAFCET-Aktionen, deren Verhalten nicht direkt in SFC überführt werden kann. Auch hier wird das durch sie spezifizierte dynamische Verhalten mit Hilfe von ST-Anweisungen und der Verwendung von Instanzen

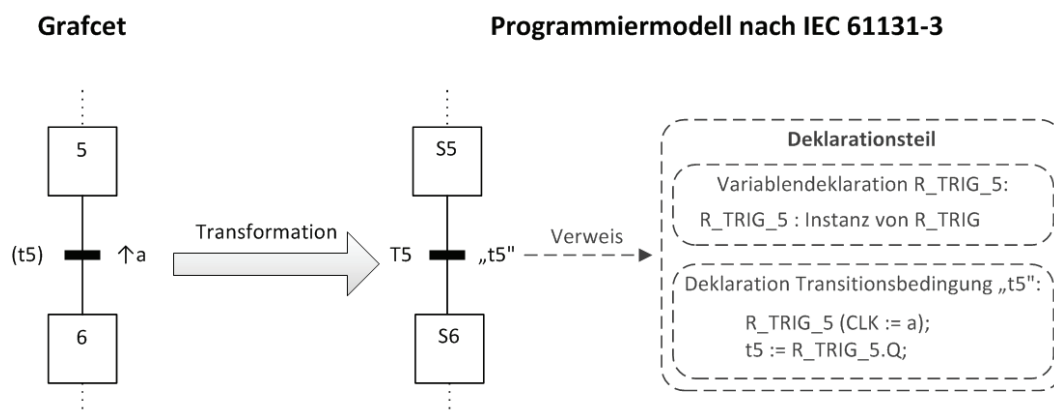


Abbildung 8-8: Transformation eines GRAFCET-Ereignisses in das Programmiermodell nach IEC 61131-3 (Modellebene)

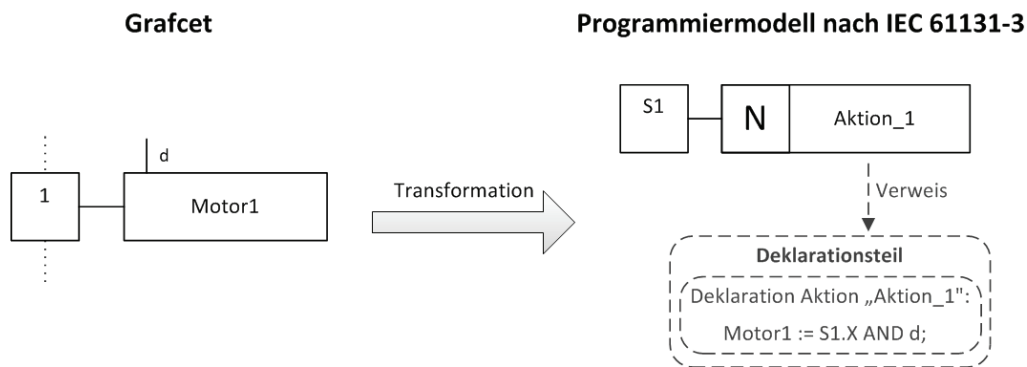


Abbildung 8-9: Transformation einer GRAFCET-Aktion mit Zuweisungsbedingung in das Programmiermodell nach IEC 61131-3 (Modellebene)

standardisierter Funktionsbausteine in das Programmiermodell der IEC 61131-3 überführt. Zusätzlich müssen noch die geeigneten Aktionsqualifier festgelegt werden, wobei in allen Fällen nur diejenigen Aktionsqualifier (N , S , R , L , D) verwendet werden, die konform zu den Bestimmungen für implementierungsunabhängige SFCs gemäß [BAUER 2003] sind. Abbildung 8-9 zeigt in diesem Zusammenhang die Transformationsregel 12 (\rightarrow Anhang D) für eine *kontinuierlich wirkende Aktion mit Zuweisungsbedingung*.

8.3.2 Transformationsregeln auf Metamodellebene

Die Transformationsregeln auf der Metamodellebene definieren, wie die im vorangegangenen Kapitel erläuterten GRAFCET-Konstrukte von ihrer PNML-Repräsentation (\rightarrow 7.3) in XML-Konstrukte überführt werden, die im Sinne des *PLCopenXML*-Schemas gültig sind. Sie betreffen somit die *Transformation 3* (\rightarrow Abbildung 8-5), wobei vorausgesetzt wird, dass die GRAFCET-Informationen in einer aus struktureller Sicht sowie hinsichtlich der Initialschritte normalisierten PNML-Datei vorliegen. Ausgehend von dieser Annahme handelt es sich bei den Regeln für die Metamodellebene um eine textuelle Repräsentation der Transformationsregeln auf Modellebene, die einer spezifischen XML-Syntax angepasst sind. Anhand von zwei Beispielen sollen die hierfür wesentlichen Zusammenhänge nachfolgend beschrieben werden. Eine vollständige Übersicht über die Transformationsregeln auf Modellebene ist in Anhang C dieser Arbeit beigelegt. Im Allgemeinen können die Transformationsregeln untergliedert werden in einen Teil von Regeln, die zusätzliche Angaben im Deklarationsteil eines Programms erfordern, wie beispielsweise Transitionsbedingungen und Aktionen ohne direkte Entsprechung, und einen Teil von Regeln, die keine zusätzlichen Angaben im Deklarationsteil erfordern. Zu Letztgenannten gehört beispielsweise die Vorschrift zur Überführung einer *kontinuierlich wirkenden Aktion*, wie für die Modellebene am Beispiel aus Abbildung 8-7 demonstriert. Die entsprechende Transformationsvorschrift für die Metamodellebene ist in Abbildung 8-10 aufgezeigt, in der offensichtlich wird, dass PNML und *PLCopenXML* unterschiedliche Strategien verfolgen, um den Informationsgehalt eines Grafcet abzuspeichern.

In der PNML stehen Schritte, Transitionen und Wirkverbindungen als zentrale Netzelemente eines Petrinetzes im Vordergrund. Transitionsbedingungen werden innerhalb der *transition*-Elemente und Aktionen innerhalb der *place*-Elemente beschrieben und als werkzeugspezifische Informationen gekennzeichnet (*toolspecific*). Auch die zugehörigen Ein- und Ausgangsvariablen werden innerhalb dieser Elemente deklariert. Insbesondere bei einer

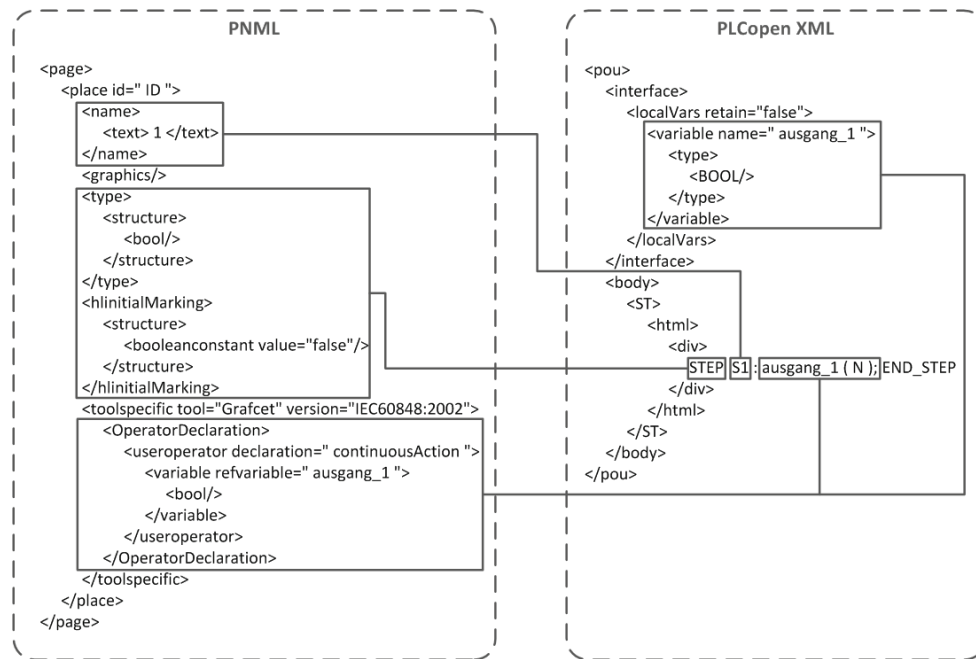


Abbildung 8-10: Transformation einer kontinuierlich wirkenden Aktion in Steuerungscode gemäß *PLCopenXML* (Metamodellebene)

mehrfachen Verwendung einer Variablen innerhalb des Grafcet kann es dadurch zu einer entsprechend mehrfachen Variablendeklaration im PNML-Dokument kommen. *PLCopenXML* orientiert sich hingegen vorrangig an der in IEC 61131-3 definierten Programmstruktur, bestehend aus Deklarations- und Anweisungsteil. Somit ist die Deklaration von Variablen, Aktionen und Transitionsbedingungen grundsätzlich getrennt von der Beschreibung der Netzstruktur. Aus der Perspektive der Transformation müssen die in der PNML-Datei enthaltenen Daten zunächst sortiert und anschließend auf den Deklarations- und Anweisungsteil der *PLCopenXML*-Datei verteilt werden. Wenn ein GRAFCET-Konstrukt durch eine direkte Entsprechung in das Programmiermodell der IEC 61131-3 abgebildet werden kann, müssen keine zusätzlichen Konstrukte in der *PLCopenXML*-Datei deklariert werden. Ist eine direkte Entsprechung nicht möglich, so sind zusätzliche Variablen notwendig, wie am Beispiel eines GRAFCET-Ereignisses in Abbildung 8-11 veranschaulicht wird. In diesem Beispiel muss zusätzlich eine Instanz des Bausteins `R_TRIG` im Variablendeklarationsteil eingefügt werden, um das durch die steigende Flanke (*risingEdge*) spezifizierte dynamische Verhalten der Transitionsbedingung äquivalent abbilden zu können. Die in diesem und in dem vorangegangenen Unterabschnitt definierten Transformationsregeln auf Modell- und Metamodellebene basieren auf dem formalen SIPN-Modell für GRAFCET und bilden eine implementierungsunabhängige Vorschrift für die eindeutige Transformation eines Grafcet in Steuerungscode nach IEC 61131-3. Die Fortführung des somit begründeten Transformationskonzepts erfolgt im Rahmen der Ausführungen zu einer prototypischen Implementierung im nächsten Kapitel.

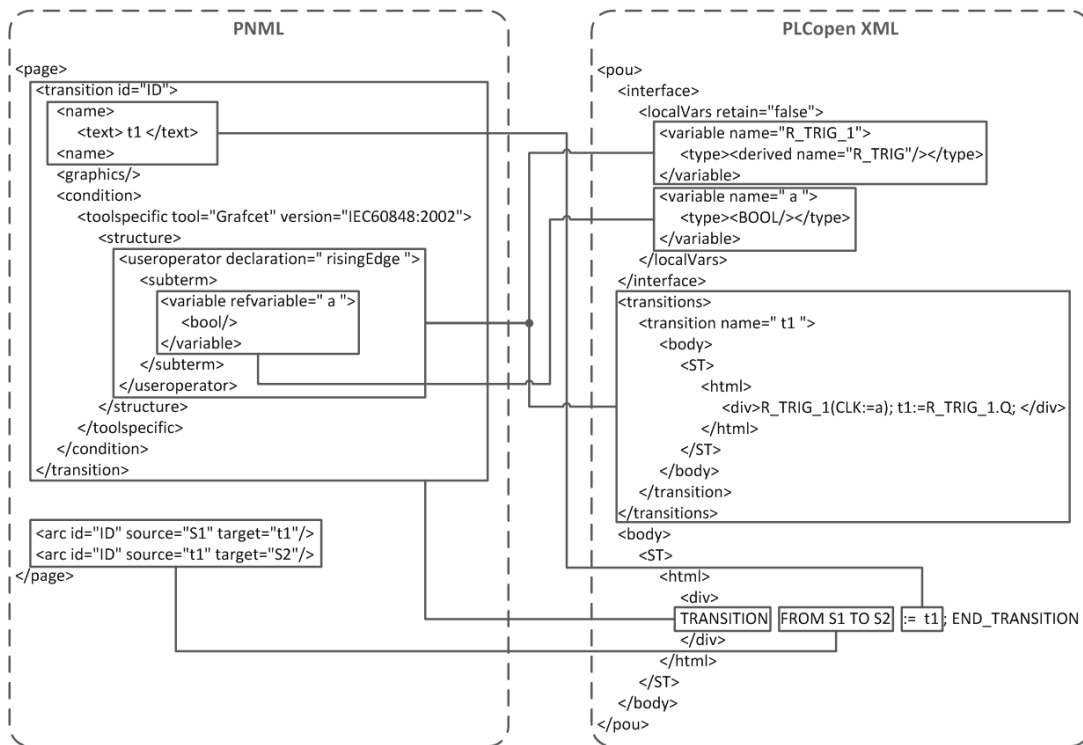


Abbildung 8-11: Transformation eines GRAFCET-Ereignisses in Steuerungscode gemäß *PLCopenXML* (Metamodellebene)

9 Werkzeugunterstützung

9.1 Werkzeug zur Erstellung von GRAFCET-Spezifikationen

9.1.1 Struktureller Aufbau und Funktionsweise

Wie im Verlauf dieser Arbeit bereits diskutiert wurde, sind aus heutiger Sicht nur wenige Werkzeuge verfügbar, die das Erstellen eines Grafcet sowie die Simulation seines dynamischen Verhaltens nach den Vorgaben der IEC 60848 ermöglichen. Die untersuchten Werkzeuge (GRAFCET-Editoren) speichern die Informationen bezüglich eines Grafcet in proprietären Datenformaten, deren Struktur und Inhalt in den überwiegenden Fällen nicht offen zugänglich ist und eine Integration der Spezifikation des Steuerungsablaufs in einen systematischen, rechnergestützten Steuerungsentwurfsprozess verhindert. Die in den vorangegangenen Kapiteln vorgestellte Methode zur automatischen Generierung IEC 61131-3 konformen Steuerungscode kann auf Basis der momentan verfügbaren GRAFCET-Werkzeuge somit nicht umgesetzt werden. Zwar zeigen beispielsweise die Autoren in [ALVAREZ ET AL. 2012] eine Methode auf, wie ausgehend von einem in *SFCedit* erstellten Grafcet automatisiert Steuerungscode gemäß IEC 61131-3 generiert werden kann. Allerdings beruht diese Methode auf einer aufwändigen experimentellen Analyse des proprietären, XML-basierten Datenformats von *SFCedit* und einem näherungsweise bestimmten XML-Schema.

Hinsichtlich einer im Rahmen der vorliegenden Arbeit geeigneten Werkzeugunterstützung war es daher zunächst notwendig, einen geeigneten GRAFCET-Editor zu entwickeln. Dieser sollte es dem Anwender erlauben, einen Grafcet gemäß IEC 60848 zu erstellen und in einem Datenformat abzuspeichern, welches eindeutig dokumentiert und für weiterführende Werkzeuge zugänglich ist. Eine integrierte Simulation des dynamischen Verhaltens lag nicht im Fokus dieser Arbeit. Im Gegensatz zu den bestehenden proprietären Software-Lösungen, wie beispielsweise *WinErs GRAFCET* oder *SFCedit*, sollte für den in dieser Arbeit vorgestellten GRAFCET-Editor möglichst eine bestehende Werkzeug-Infrastruktur genutzt werden, um die Integration anwenderseitig mit wenig Aufwand realisieren zu können.

Mit *Microsoft® Visio (MS Visio)* bietet die im Bereich der Büroanwendungssoftware weit verbreitete *Microsoft® Office* Produktfamilie [MICROSOFT]® diesbezüglich eine geeignete Plattform für die Modellierung und Visualisierung von Graphen und Plänen unter der Nutzung vordefinierter grafischer Elemente [VISIO]®. Wie am Beispiel der Formalisierten Prozessbeschreibung in [CHRISTIANSEN ET AL. 2011] aufgezeigt, eignet sich *MS Visio* durch die Möglichkeit zur Implementierung anwendungsspezifischer Elementbibliotheken auch grundsätzlich für die Modellierung von Planungsdokumenten im Engineering der Automatisierungstechnik. Eine einheitliche grafische Benutzeroberfläche ermöglicht Anwendern einen leichten Zugang zur Modellierung aufgrund einer vertrauten Arbeitsumgebung. Die nachfolgend beschriebene prototypische Implementierung des GRAFCET-Editors wird daher in Form einer anwendungsspezifischen Elementbibliothek (*Shape-Galerie*) in *MS Visio* umgesetzt.

Das spezifische Objektmodell von *MS Visio* und die davon abgeleitete parametrische Datenstruktur unterscheiden sich allerdings von gängigen *Microsoft® Office*-Produkten, wie beispielsweise *MS Word* oder *MS-Excel*, und müssen im Zuge der Implementierung eines GRAFCET-Editors berücksichtigt werden. Einen Auszug aus dem Objektmodell von

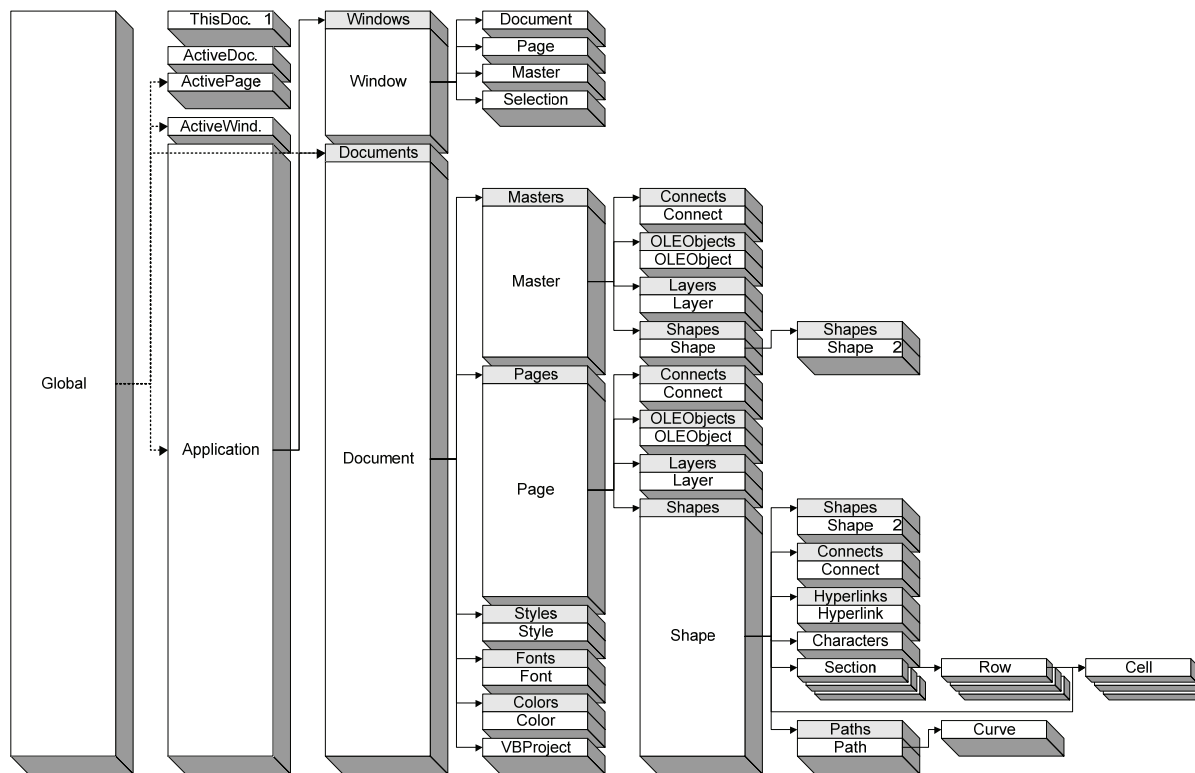


Abbildung 9-1: Auszug aus dem Objektmodell von *MS Visio*, in Anlehnung an [Martin 2011]

MS Visio, welches beispielsweise in [MARTIN 2011] ausführlich erörtert wird, zeigt Abbildung 9-1. Demnach ist jedem grafischen Element einer *MS Visio*-Elementbibliothek (*Shape*) ein *ShapeSheet* zugeordnet, welches den Großteil der elementspezifischen Eigenschaften und Attribute wie Geometrie, Farbe oder Verhalten enthält. Jede Zelle des tabellarisch aufgebauten *ShapeSheets* stellt ein Objekt des Objektmodells dar und kann entsprechend adressiert werden. Die hierarchische Gliederung des Objektmodells erfolgt von der Anwendungsebene (*Application*) zur untergeordneten Dateiebene (*Document*) und weiter zur Zeichenblattebene (*Page*). Der Zeichenblattebene wiederum untergeordnet sind die Elementebene (*Shape*) sowie die ShapeSheet-Ebene (*Cell*). Jedes Shape verweist auf eine übergeordnete, eindeutig identifizierbare Shape-Klasse, das sogenannte *MasterShape*. Des Weiteren lassen sich in einzelnen vordefinierten und nutzerspezifischen Zellen des ShapeSheets verschiedene Ereignisse definieren, bei deren Auftreten bestimmte Programm-routinen (*Makros*) aufgerufen werden. So kann beispielsweise ein Doppelklick auf ein grafisches Element das Öffnen eines Dialogfensters zur Eingabe weiterführender Daten hervorrufen. Die einem Shape auf diese Weise zugeordneten Methoden werden mit Hilfe der auf *Office*-Anwendungen zugeschnittenen Programmiersprache *Visual Basic for Applications* (*VBA*) implementiert und innerhalb des *ShapeSheets* referenziert.

Die Daten eines *MS Visio*-Elements sind im ShapeSheet abgelegt und werden durch das zugeordnete Shape grafisch repräsentiert (→ Abbildung 9-2). Einerseits sind die im ShapeSheet hinterlegten Informationen auf das MasterShape zurückzuführen, wie beispielsweise Informationen bezüglich der geometrischen Form und Farbgebung. Andererseits werden Shape-spezifische Informationen bei Instanziierung auf dem Zeichenblatt oder weiteren Ereignissen durch entsprechende VBA-Makros erzeugt und im ShapeSheet abgespeichert. Die für ein Shape implementierten VBA-Makros werden in einem *Modul*

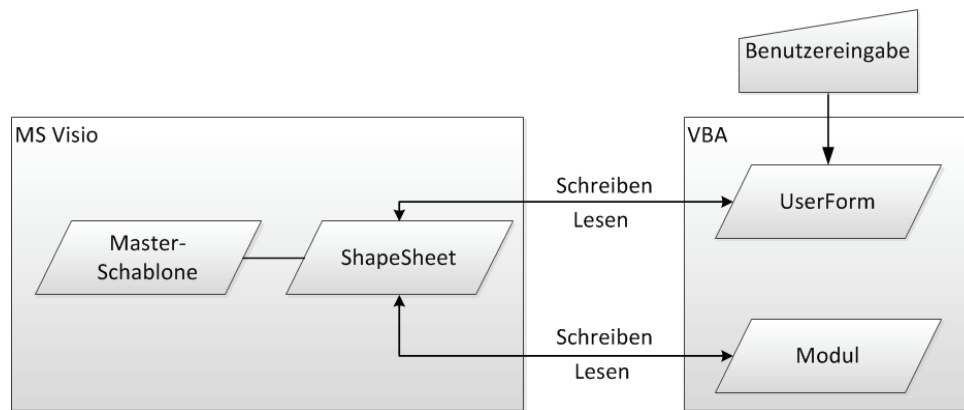


Abbildung 9-2: Integration von VBA und ShapeSheet in MS Visio

zusammengefasst. Darüber hinaus besteht für den Anwender die Möglichkeit, verschiedene Attribute über einen Benutzerdialog einzugeben und zu beeinflussen. Die Implementierung der grafischen Repräsentation und Funktionalität des Benutzerdialogs ist ebenfalls Shape-spezifisch und erfolgt aus VBA heraus in der sogenannten *UserForm*. Somit besteht die Möglichkeit, Anwendungen für *MS Visio* in Form nutzerspezifischer Shape-Galerien nahezu beliebig zu erstellen oder anzupassen. Eine ebensolche nutzerspezifische Shape-Galerie stellt der hier vorgestellte GRAFCET-Editor dar. So sind alle GRAFCET-spezifischen Elemente, wie beispielsweise Schritte, Transitionen und Aktionen, in Form von Shapes in einer Shape-Galerie gebündelt und werden in einem spezifischen Dateiformat abgespeichert. Die Shape-Galerie ist somit eine Bibliothek, die in die *MS Visio*-Umgebung eingebunden werden kann. Durch das *drag&drop* Verfahren können Instanzen der einzelnen Shapes auf dem Zeichenblatt positioniert und individuelle Parameter und Eigenschaften durch einen über Rechtsklick erreichbaren Nutzerdialog definiert werden.

Eine automatische Verbindungsfunktion sowie weitere Standardfunktionen von *MS Visio* erleichtern die Erstellung und Anordnung der Elemente eines Grafnet. Die auf den einzelnen Zeichenblättern abgelegten Shape-Instanzen bilden zusammen den Grafnet und können in verschiedenen Dateiformaten abgespeichert werden. Im Rahmen dieser Arbeit wird ausschließlich das XML-basierte Dateiformat der *MS Visio*-Zeichnung (*.vdx) berücksichtigt, welches die Shape-Daten enthält und gemäß dem Objektmodell strukturiert ist (*Visio-XML-Format*).

9.1.2 Erstellung einer GRAFCET-Spezifikation

Die Spezifikation eines Steuerungsablaufs mit Hilfe des GRAFCET-Editors erfolgt aus *MS Visio* heraus durch das Einbinden der GRAFCET Shape-Galerie und die Aktivierung der zugehörigen Makro-Inhalte. In der Bedienoberfläche des GRAFCET-Editors werden die verfügbaren grafischen Elemente gemäß IEC 60848 dann als Piktogramme am linken Bildrand angezeigt und können durch *drag&drop* in den Arbeitsbereich gezogen werden (→ Abbildung 9-3). Jedes grafische Element des Grafnet erhält eine eindeutige Identifikationsnummer. Weitere Parameter, wie beispielsweise der Initialzustand von Schritten oder die Definition einer Transitionsbedingung, können anschließend über ein elementspezifisches Auswahlmü und damit verknüpfte *UserForms* spezifiziert werden. Hinsichtlich der Spezifikation von Transitionsbedingungen und Aktionen ermöglichen die entsprechenden *UserForms* den Import von Daten aus einer *MS Excel*-Tabelle über eine von

MS Visio bereitgestellte Dateischnittstelle. So können Daten aus einer *MS Excel*-Tabelle, wie beispielsweise eine Auflistung von Variablen und deren charakteristischen Eigenschaften (*Signalliste*), in den GRAFCET-Editor importiert und für die Spezifikation des Steuerungsablaufs genutzt werden. Der zugehörige Import-Algorithmus des GRAFCET-Editors ist auf eine einfache Signalliste abgestimmt und sortiert die Tabelleneinträge nach Datentyp (Boolesch oder numerisch) und Signalart (Eingangs- oder Ausgangssignal). Die als Tabelleneinträge importierten Variablen werden dann innerhalb der *UserForms* zur Spezifikation von Transitionsbedingungen und Aktionen für den Anwender gefiltert und in einem Auswahlménú zur Verfügung gestellt. Zusätzlich steht es dem Anwender frei, eigene, neue Variablen innerhalb des Grafcet zu verwenden. Sofern gewünscht, können diese der Signalliste hinzugefügt werden. Der Import-Algorithmus ist in Abhängigkeit des konkreten Anwendungsfalls oder der Beschaffenheit der konkreten Signalliste modifizierbar, so dass eine individuelle Anpassung des GRAFCET-Editors möglich ist.

Der Arbeitsbereich des GRAFCET-Editors zeigt das momentan aktive Zeichenblatt, dessen Format dynamisch erweitert wird, sobald die durch den Grafcet beanspruchte Zeichenfläche die aktuellen Abmessungen überschreitet. Insofern können grundsätzlich beliebig umfangreiche Grafcets auf einem Zeichenblatt erstellt werden. Die aus *MS Office* bekannten Ansichtsfunktionen, wie beispielsweise *Zoom in* oder *Zoom out*, sind auch im GRAFCET-Editor verfügbar. Das Zeichenblatt-Register am unteren Rand des Arbeitsbereichs zeigt die einzelnen Teil-Grafcets (G^*), Makroschritt-Feinstrukturen (M^*) und Teil-Grafcets der eingeschlossenen Schritte ($X^*/G\#$) des Grafcets. Jeder dieser Teil-Grafcets wird in einem gesonderten Zeichenblatt spezifiziert. Die Zeichenblätter sind durch die jeweiligen Namen gemäß IEC 60848 eindeutig gekennzeichnet und erlauben eine eindeutige Zuordnung zu den übergeordneten Elementen, wie beispielsweise Makroschritten oder *einschließenden Schritten*. Der allen Teil-Grafcets übergeordnete globale Grafcet wird auf dem Zeichenblatt mit Namen *Global-Grafcet* spezifiziert und ist bei Start des GRAFCET-Editors

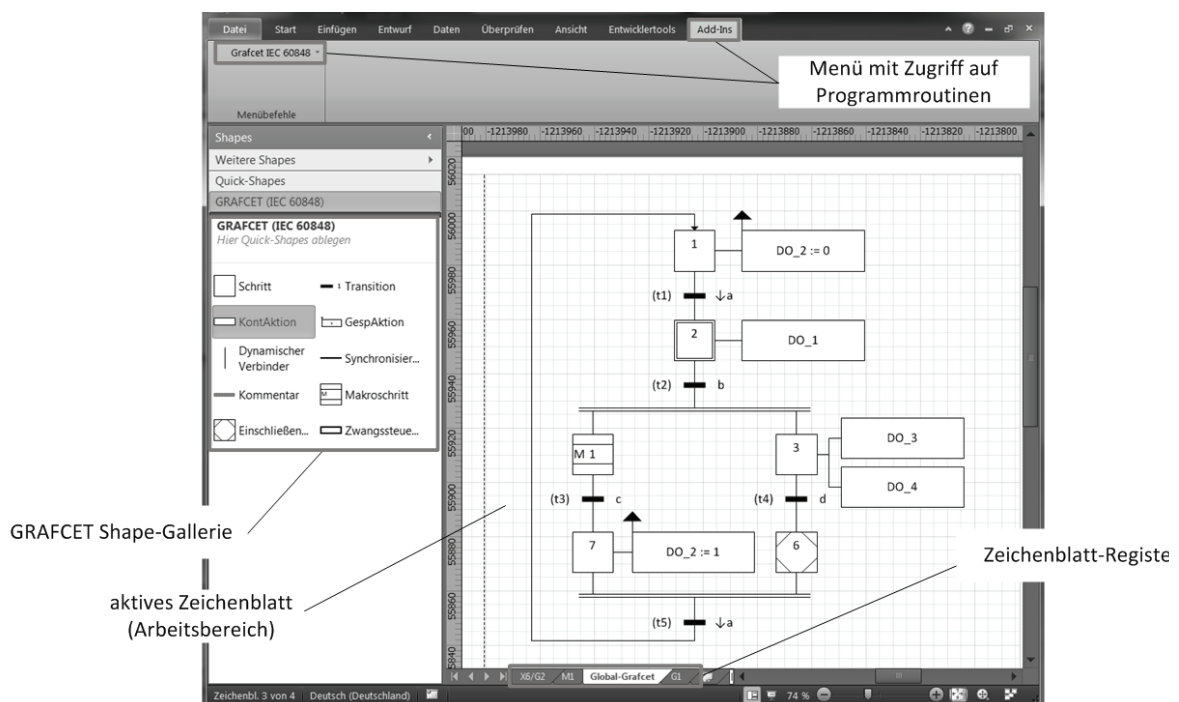


Abbildung 9-3: Bedienoberfläche des GRAFCET-Editors in *MS Visio*

das einzig vorhandene Zeichenblatt. Sofern ein Grafcet keine hierarchische Struktur aufweist, besteht er nur aus einem globalen Grafcet.

Grundsätzlich ermöglicht der GRAFCET-Editor das Erstellen beliebiger, auch unvollständiger oder gar fehlerhafter Grafkets im Sinne der IEC 60848. Somit können auch Zwischenergebnisse einer Spezifikation dokumentiert werden. Grundsätzlich ist der Anwender dafür verantwortlich, dass es sich bei dem von ihm spezifizierten Grafcet um einen gültigen Grafcet (→ 6.1.1) handelt. Zur Unterstützung des Anwenders stehen hierfür im GRAFCET-Editor verschiedene, auf GRAFCET angepasste Prüfroutinen zur Verfügung, die als *Add-Ins* manuell ausgewählt und gestartet werden können. Ein häufig auftretender Fehler bei der Erstellung eines Grafcet in der Umgebung von *MS Visio* ist beispielsweise, dass Anfangs- und Endpunkte von Wirkverbindungen nicht exakt mit den Verbindungspunkten der Schritt- und Transitionselemente übereinstimmen. Für eine nachfolgende rechnergestützte Auswertung und Transformation ist dies allerdings erforderlich. Eine in der momentanen Version des GRAFCET-Editors auswählbare Prüfroutine bietet daher eine automatisierte Analyse der Wirkverbindungen und zeigt fehlerhafte Wirkverbindungen grafisch an. Eine weitere Prüfroutine analysiert, ob stets nur diejenigen GRAFCET-Elemente miteinander verbunden sind, die gemäß IEC 60848 miteinander verbunden sein dürfen. Aufgrund ihres modularen Aufbaus sind auch weitere Prüfroutinen im Zuge einer Weiterentwicklung des GRAFCET-Editors denkbar.

Wurde ein gültiger Grafcet erstellt und zuvor mit den verfügbaren Prüfroutinen analysiert, so stehen in *MS Visio* unterschiedliche Dateiformate für die Speicherung der Daten zur Verfügung. Das im Rahmen dieser Arbeit relevante Dateiformat ist das *Visio-XML-Format* (→ Abbildung 9-4). Ein *MS Visio* XML-Dokument besteht demnach aus Angaben zur Datei, wie beispielsweise Namen, Titel, Autor oder Zeitstempel. Diese Dokumentinformationen

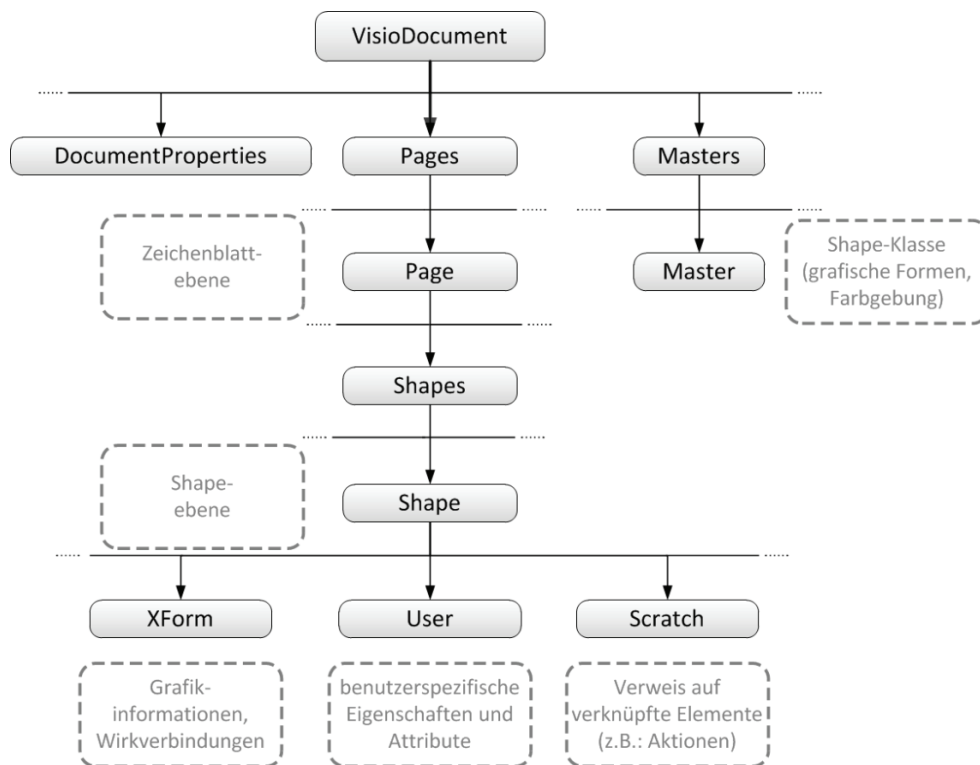


Abbildung 9-4: Grundlegende Struktur des *Visio-XML-Formats*

werden im Element *DocumentProperties* abgelegt. Die einzelnen Shape-Klassen werden als *Master*-Elemente gespeichert und definieren die in der Shape-Galerie verfügbaren Grafikelemente. Die Inhalte der einzelnen Zeichenblätter eines *MS Visio*-Dokuments werden in den *Page*-Elementen durch die vorhandenen Shapes beschrieben. Jedes *Shape*-Element enthält die im ShapeSheet enthaltenen Daten, jedoch keine direkte Information darüber, um welches Grafikelement es sich handelt. Das Shape-Element verweist lediglich auf das zugehörige *Master*-Element. Neben den für Graficet relevanten Shape-Informationen in den Elementen *XForm*, *User* und *Scratch* enthält das Shape-Element weitere grafische Daten, die im Rahmen der in dieser Arbeit vorgestellten Methode zur automatischen Generierung von IEC 61131-3 konformem Steuerungscode allerdings nicht relevant sind.

Zum Zeitpunkt der Erstellung dieser Arbeit ermöglicht der GRAFCET-Editor die Erstellung umfangreicher, insbesondere durch Makroschritte, *einschließende Schritte* und *zwangssteuernde Befehle* hierarchisch strukturierter Graficets. Bezüglich des Wirkungsteils werden hinsichtlich der Aktionen alle gemäß IEC 60848 möglichen und im Rahmen dieser Arbeit gültigen Konstrukte berücksichtigt. Lediglich die Spezifikation zeitabhängiger Transitionsbedingungen wird durch die aktuelle Version nicht unterstützt. Eine Ergänzung zeitabhängiger Transitionsbedingungen wird allerdings für zukünftige Versionen des GRAFCET-Editors angestrebt.

9.2 Werkzeug für die Transformation

9.2.1 Struktureller Aufbau und Funktionsweise

Nachdem im vorangegangenen Kapitel mit dem auf *MS Visio* basierenden GRAFCET-Editor die Idee und Implementierung eines Werkzeugs zur Erstellung von IEC 60848 konformen Steuerungsspezifikationen vorgestellt wurde, wird in diesem Kapitel beschrieben, wie darauf aufbauend das in Abbildung 7-5 veranschaulichte Transformationskonzept werkzeugtechnisch unterstützt werden kann. Ein Graficet wird in dem werkzeugspezifischen, XML-basierten Datenformat des GRAFCET-Editors abgespeichert und bildet den Ausgangspunkt für die nachfolgenden Transformationsschritte. Durch geeignete Algorithmen sollen die GRAFCET-relevanten Informationen in einem ersten Transformationsschritt zunächst extrahiert und, den GRAFCET-spezifischen Abbildungsvorschriften (→ 8.3) folgend, im werkzeugunabhängigen PNML-Format abgelegt werden. Nach der sich anschließenden Normalisierung der Struktur sowie der Initialschritte sollen in einem abschließenden Transformationsschritt die im PNML-Format vorliegenden GRAFCET-Informationen durch geeignete Algorithmen systematisch in das *PLCopenXML*-Format überführt werden.

Aus der Perspektive der zugehörigen XML-basierten Datenformate, die gemäß des *MS Visio* XML-, des PNML- oder des *PLCopenXML*-Schemas gültig sind, müssen bestimmte, in der jeweiligen XML-Struktur vorhandene Elemente und Attribute in Elemente und Attribute einer anderen XML-Struktur übersetzt werden. Die zugrunde liegenden XML-Schemata sind frei verfügbar und somit bekannt. Die Familie der XML-Technologien bietet mit der *eXtensible Stylesheet Language Transformations* (XSLT) [W3C XSLT 2.0][#] eine XML-basierte Beschreibungssprache für Abbildungsregeln von XML-Dokumenten und könnte somit im Zuge dieser Arbeit genutzt werden. Ein sogenannter XML-Parser, der als Werkzeug für die Transformation dient, würde das Ursprungsdokument einlesen, die im XSLT-Stylesheet dokumentierten Termersetzungsvorschriften (*Templates*) darauf anwenden und ein

entsprechendes Ausgabedokument erzeugen. Somit könnte im Rahmen der Transformation auf bewährte, marktübliche XML-Werkzeuge zurückgegriffen werden. Es müssten lediglich die Transformationsregeln in Form von XSLT-Stylesheets erstellt werden. Ein wesentlicher Nachteil von XSLT ist allerdings, dass komplexe Transformationen nur umständlich in einem Stylesheet beschrieben werden können, wie beispielsweise einfache Schleifenausdrücke durch Rekursion. Zudem ist es nicht möglich, Zwischenergebnisse der einzelnen XSLT-Templates zu speichern und in weiteren Templates zu verwenden. Im Falle des *MS Visio* XML-Formats würde dies beispielsweise bedeuten, dass die Verknüpfung eines Shape-Elements mit seinem zugehörigen Master-Element nur mit sehr großem Aufwand oder gar nicht realisierbar ist. Sind die Daten bezüglich eines GRAFCET-Elements im XML-Dokument verteilt, so können sie im Zuge der Transformation nur schwer zusammengeführt werden. Die Transformationsregeln zur Normalisierung von Struktur und Initialschritten könnten ebenfalls nur mit großem Aufwand in einem Stylesheet hinterlegt werden. Aufgrund der zahlreichen Gestaltungsmöglichkeiten und Zusammensetzung eines Grafcet wären die zugehörigen Stylesheets entsprechend umfangreich und nur schwer modifizierbar. Hinsichtlich der Implementierung der konzeptionellen Anforderungen an die Transformation eines Grafcet wird XSLT demzufolge als ungeeignet bewertet.

Der GRAFCET-Transformator wird aus diesem Grund als integrierte Software-Plattform, in der der Anwender das Ursprungsdokument sowie das gewünschte XML-Format der Ausgabedatei manuell auswählen kann, entwickelt und implementiert. Die einzelnen, zur Erzeugung der Ausgabedatei notwendigen Transformationsschritte werden durch die zugehörigen Transformationsalgorithmen automatisch durchgeführt und die Ausgabedatei anschließend abgespeichert (→ Abbildung 9-5). Die interne Struktur des Transformationswerkzeugs beruht auf einem GRAFCET-spezifischen temporären Objektmodell, welches die aus einer *MS Visio* XML-Datei extrahierten Daten importiert. Abhängig von dem durch die Benutzereingabe ausgewählten Ausgabeformat der Transformation werden die Daten des internen Objektmodells anschließend bearbeitet und ausgegeben. Sowohl die Abbildung der GRAFCET-spezifischen Daten aus der *MS Visio* XML-Datei in das interne Objektmodell als auch die Abbildungen der Daten des internen Objektmodells auf die Ausgabe-XML-Formate werden, unter Berücksichtigung der entsprechenden Transformationsregeln, durch eine *Mapping*-Funktion realisiert. Nach Abschluss einer Transformation wird das interne Objektmodell verworfen und erst dann wieder zur Laufzeit des Transformationsalgorithmus erzeugt, wenn eine neue Transformation gestartet wird. Das interne Objektmodell dient somit als „Vehikel“ für die Transformation.

Die prototypische Implementierung des GRAFCET-Transformators erfolgte in der Entwicklungsplattform *Microsoft Visual Studio* unter Nutzung der objektorientierten Programmiersprache *C#* [KÜHNEL 2010]. *C#* bietet verschiedene Schnittstellen für das Einlesen von XML-Dateien, wie beispielsweise die in der Klassenbibliothek *System.Xml* verfügbare Klasse *XMLReader*, welche im Rahmen des vorgestellten Transformationswerkzeugs verwendet wird. Eine innerhalb des Transformationsalgorithmus erzeugte Instanz der Klasse *XmlReader* ermöglicht das Einlesen eines XML-Dokuments als Stream, was bedeutet, dass die XML-Elemente und -Attribute einzeln und nacheinander gelesen werden (Vorwärtszugriff). Im Gegensatz zu anderen XML-Schnittstellen, wie beispielsweise dem *Document Objekt Model* (DOM), ist der Zugriff auf die eingelesenen Daten bei Verwendung

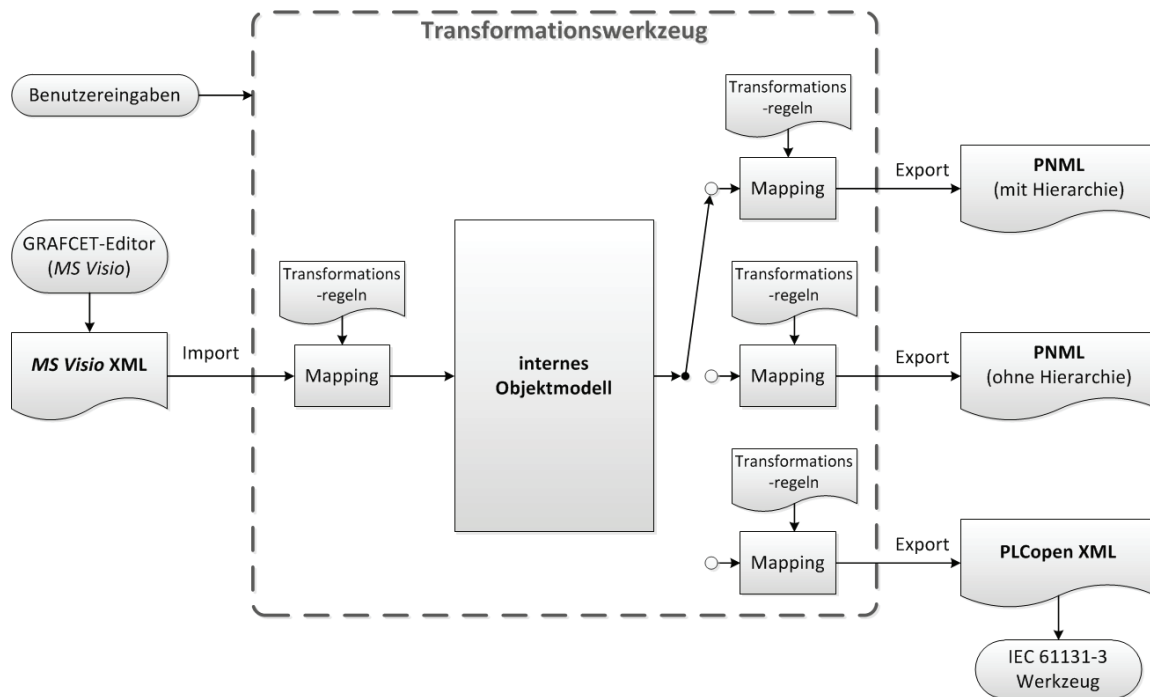


Abbildung 9-5: Aufbau und interne Struktur des Transformationswerkzeugs

der Klasse *XmlReader* nicht unmittelbar möglich. Sie müssen zunächst durch ein Mapping gemäß den Abbildungsregeln in dem internen Objektmodell zwischengespeichert werden. Da das interne Objektmodell einen einfachen Zugang für nachfolgende Transformationsschritte ermöglicht und integraler Bestandteil des GRAFCET-Transformators ist, ist ein Mapping sowohl bei Verwendung der Klasse *XmlReader* als auch unter Nutzung der DOM-Schnittstelle notwendig. Die Vorteile beider Methoden sind dadurch nicht eindeutig voneinander abgrenzbar. Insofern ist im Rahmen der hier vorgestellten Implementierung die Wahl zugunsten der Klasse *XmlReader* eine rein subjektive Entscheidung.

Hinsichtlich des Exports der transformierten GRAFCET-spezifischen Daten in die Ausgabeformate PNML und *PLCopenXML* stehen in C# die gleichen Schnittstellen für das Schreiben von XML-Dokumenten zur Verfügung. Als Gegenstück zur Klasse *XmlReader* stellt die Klassenbibliothek *System.Xml* die Klasse *XmlWriter* zur Verfügung, die ein XML-Dokument in Form eines Streams erstellt. Einer Instanz der Klasse *XmlWriter* müssen dazu die relevanten Daten genau dann übergeben werden, wenn sie in dem XML-Dokument geschrieben werden sollen. Daraus ergibt sich eine vergleichsweise komplizierte Implementierung für die Generierung der entsprechenden XML-Struktur [Kühnel 2010]. Eine ähnliche Schnittstelle für das Bearbeiten und Erstellen von XML-basierten Datenaustauschformaten auf der Grundlage des internen Objektmodells bietet die Klassenbibliothek *System.Xml.Linq*. Sie nutzt die auf der Microsoft Software-Plattform *.Net* integrierte Abfragesprache *Language INtegrated Query* (LINQ) [LINQ][®]. Eine ausführliche, auf Automatisierungstechnische Zwecke bezogene Diskussion von *LINQ to XML* findet sich beispielsweise in [Barth & Fay 2010]. Die XML-Struktur des Ausgabedokuments wird durch Instanzen der Klasse *XStreamingElement* beschrieben. Über entsprechende LINQ-Abfragen kann auf die notwendigen Daten des internen Objektmodells zugegriffen werden.



Abbildung 9-6: Schematischer Ablauf des Transformationsalgorithmus

Im Softwaremodell des Transformationsalgorithmus wird das Einlesen des *MS Visio*-spezifischen XML-Dokuments und das Mapping in das interne Objektmodell in der Klasse *Reader* gekapselt (→ Abbildung 9-6). Nachdem die GRAFCET-spezifischen Daten vollständig extrahiert und in dem internen Objektmodell zwischengespeichert sind, müssen sie gemäß den Anforderungen des jeweiligen Ausgabeformats bearbeitet werden. Als Ausgabeformate für die importierten GRAFCET-Daten sind die PNML sowie das *PLCopenXML*-Format vorgesehen. Bezüglich der Transformation in die XML-Struktur der PNML besteht entweder die Möglichkeit, die hierarchische Struktur des Grafcet (*PNML*) oder die normalisierte Struktur des Grafcet (*NPNML*) in der PNML-Struktur abzubilden. Die zugehörige Mapping-Funktionalität zwischen internem Objektmodell und dem Ausgabeformat wird im Softwaremodell durch die Klassen *PNMLWriter*, *NPNMLWriter* und *PLCopenXMLWriter* bereitgestellt. Vor der Generierung eines NPNML-Dokuments müssen die konzeptionell festgelegten Transformationsschritte zur Normalisierung der Struktur durchgeführt werden. Die zugehörigen Methoden für diese Modifizierung des internen Objektmodells sind in der Klasse *ConverterN* des Softwaremodells enthalten. Im Falle von *PLCopenXML* ist die zusätzlich notwendige Normalisierung der Initialschritte in die Klasse *PLCopenXMLWriter* integriert. Die jeweiligen Transformationsregeln sind in den Klassen *PNMLWriter*, *NPNMLWriter* und *PLCopenXMLWriter* als *case-Abfragen* in den jeweiligen Transformationsalgorithmus integriert. Sie werden auf die Daten des internen Objektmodells angewendet.

9.2.2 Durchführung der Transformation

Die für den Anwender sichtbare Benutzeroberfläche des GRAFCET-Transformators öffnet sich nach Programmstart und stellt die wesentlichen nutzbaren Funktionen dar (→ Abbildung 9-7). Die Ursprungsdatei sollte eine gemäß der *MS Visio*-Objektstruktur gültige XML-Datei sein (.vdx). Andere XML-Strukturen würden zu einer fehlerhaften Transformation führen. Die Ausgabedatei erhält automatisch die Endung *.xml*. Im Anschluss wird die gewünschte Transformation durch das einmalige Betätigen der entsprechenden Schaltfläche gestartet. Über die Statusanzeige ist für den Anwender ersichtlich, ob die Transformation erfolgreich abgeschlossen wurde oder ob Fehler aufgetreten sind. Die derzeitige Version des GRAFCET-Transformators ist auf die momentane Version des GRAFCET-Editors abgestimmt, so dass Modifikationen stets mit Hinblick auf beide Werkzeuge durchgeführt werden sollten, um fehlerhafte Transformationen oder einen vorzeitigen Programmabbruch zu vermeiden. Hinsichtlich eines systematischen Steuerungsentwurfs bietet der GRAFCET-Transformator, gemeinsam mit dem GRAFCET-Editor, eine geeignete Unterstützung des Anwenders für die Erstellung von GRAFCET-Spezifikationen gemäß IEC 60848 und die anschließende rechnergestützte Transformation in eine implementierungsunabhängige Notation (→ 7) sowie in IEC 61131-3 konformen Steuerungscode auf Basis eindeutiger Transformationsregeln (→ 8). Die dafür notwendigen Transformationsregeln beruhen auf dem formalen SIPN-Modell für GRAFCET (→ 6) und erlauben eine eindeutige automatische Interpretation der spezifizierten GRAFCET-Konstrukte.

Erste Untersuchungen in Bezug auf eine weitere Verwendung der erzeugten XML-Dateien zeigen, dass die von dem Transformationswerkzeug ausgegebenen PNML-Dateien in Werkzeugen zur Spezifikation und Analyse von Petrinetzen, wie beispielsweise *TINA* [TINA][@] oder *ePNK* [ePNK][@], importiert werden können. Der Importalgorithmus des Werkzeugs *TINA* erkennt und überspringt erwartungsgemäß die werkzeugspezifischen XML-Elemente *toolspecific*, wie im Standard ISO/IEC 15090-2 vorgesehen. Aufgrund der GRAFCET-spezifischen Abbildungsregeln in PNML wird lediglich die grundlegende Struktur eines Grafcet, bestehend aus Stellen, Transitionen und Wirkverbindungen, beim Datenimport

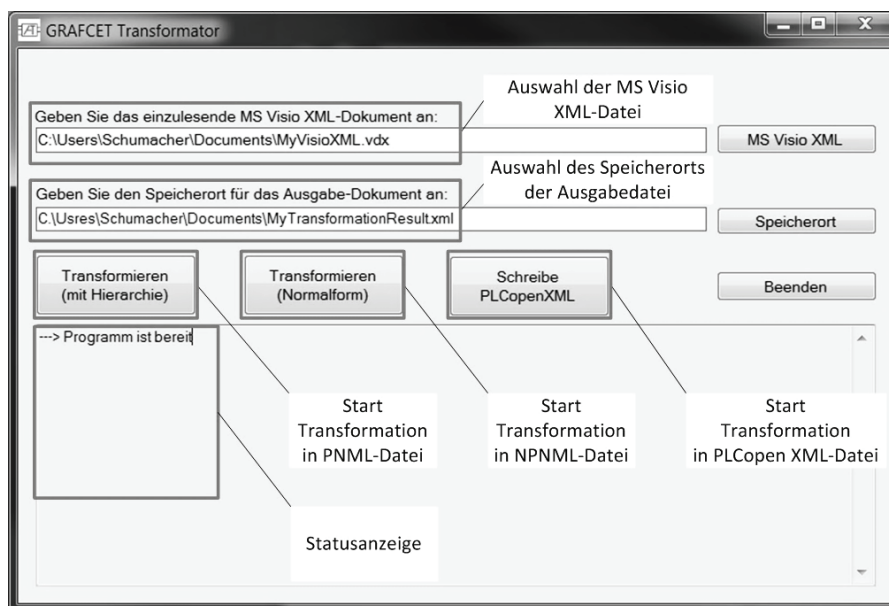


Abbildung 9-7: Bedienoberfläche des GRAFCET-Transformators

berücksichtigt und im *TINA*-Werkzeug zur Verfügung gestellt. Im Falle von *ePNK* wird der Importvorgang hingegen abgebrochen, da werkzeugspezifische Elemente nicht interpretiert werden können. Die Möglichkeit, zusätzliche Funktionalitäten in *ePNK* zu integrieren, bietet hier einen Ansatzpunkt für einen reibungslosen Import der generierten PNML-Datei.

Bezüglich des Imports einer durch den GRAFCET-Transformator erzeugten *PLCopenXML*-Datei ist anzumerken, dass die einzelnen IEC 61131-3 Programmierwerkzeuge (→ 3.6) entsprechende Import-Schnittstellen in unterschiedlichem Umfang zur Verfügung stellen (→ Tabelle 9-1). So ist der Import von *PLCopenXML*-Dateien in *STEP 7* grundsätzlich nicht möglich. Im Falle des Programmierwerkzeugs *Control Build* kann der Importvorgang zwar aufgerufen werden. Er wird aber kurz darauf abgebrochen ohne dass Daten aus dem Deklarations- und Anweisungsteil übernommen werden. Eine mögliche Ursache hierfür ist, dass das von *Control Build* unterstützte, herstellerspezifisch interpretierte *PLCopenXML*-Format von den allgemeinen Definitionen in [PLCOPEN 2009][#] abweicht. Ein ähnliches Resultat ergibt sich für *Logi.CAD*, welches lediglich die Programmiersprache FBS unterstützt. Des Weiteren ist ein Import der automatisch generierten *PLCopenXML*-Datei in *Multiprog* nur dann möglich, wenn die XML-Elemente zuvor manuell um *Multiprog*-spezifische Attribute ergänzt werden. Allerdings wird bei diesem Importvorgang nur der Deklarationsteil der *PLCopenXML*-Datei in *Multiprog* übernommen. Am Beispiel der aktuellen Version 3.5 von *CODESYS* zeigt sich, dass die durch den GRAFCET-Transformator erzeugte *PLCopenXML*-Datei zwar vollständig importiert werden kann, eine Interpretation der textuellen Repräsentation von SFCs im Anweisungsteil allerdings nicht möglich ist. Ein möglicher Grund hierfür ist, dass *CODESYS* nur die grafische Repräsentation von SFCs semantisch unterstützt.

Tabelle 9-1: Werkzeugunterstützung für Import und Export von SFCs im *PLCopenXML*-Format

Werkzeugunterstützung bezüglich automatisch generierter <i>PLCopenXML</i> -Datei	SPS-Programmierwerkzeug				
	<i>STEP 7</i>	<i>CODESYS</i>	<i>Multiprog</i>	<i>Control Build</i>	<i>Logi.CAD</i>
Import (gesamt) durchführbar	-	+	o ⁴	-	-
Import Deklarationsteil möglich	-	+	+ ⁴	-	-
Import Anweisungsteil möglich	-	+	-	-	-
Interpretation Anweisungsteil möglich	-	-	-	-	-

Legende: +: erfüllt/vorhanden/möglich
 o: teilweise erfüllt/teilweise vorhanden/teilweise möglich
 - : nicht erfüllt/nicht vorhanden/nicht möglich

⁴ zuvor manuelle Modifikation erforderlich

10 Anwendungsbeispiel

10.1 Anlagenbeschreibung und textuelle Spezifikation

Die im Rahmen dieses Kapitels betrachtete Anlage (**Prüfautomat**) dient der automatisierten mechanischen und elektrischen Funktionsprüfung von Verschlüssen für Geschirrspülmaschinen und stellt ein typisches Anwendungsbeispiel industrieller Prüfsysteme aus dem Bereich der Qualitätssicherung dar. Um den hohen Kundenanforderungen bezüglich der gewünschten Produktqualität gerecht zu werden, wird jedes vom Hersteller gefertigte Produkt (**Prüfteil**) hinsichtlich seiner mechanischen und elektrischen Funktionsfähigkeit im direkten Anschluss an den Produktionsprozess getestet. Werden alle Tests erfolgreich absolviert, so wird das Prüfteil als **Gutteil** für den Verkauf freigegeben. Nicht erfolgreich getestete Prüfteile werden repariert und als **Schlechtteile** ausgesondert. Der betrachtete Prüfautomat besteht dazu aus einer geschlossenen Prüfwelle mit Schutztüren und einem Schaltschrank mit Anzeige- und Bedienkomponente, welcher die Steuerungs-Hardware beinhaltet. Zur Steuerung stehen hardwareseitig eine SPS und eine sicherheitsgerichtete SPS (*Safety SPS*) zur Verfügung. Über die Anzeige- und Bedienkomponente werden dem Anlagenbediener einerseits Störungen und Betriebszustände in einem Text-Display angezeigt und andererseits das manuelle Starten/Stoppen der Anlage und die Umschaltung zwischen den Betriebsarten *Automatikbetrieb* und *Handbetrieb* über entsprechende Schalter/Taster ermöglicht. Im Regelfall läuft der Prüfautomat in der Betriebsart *Automatikbetrieb*. Im Falle von Störungen muss zunächst eine Betriebsartenumschaltung in den *Handbetrieb* erfolgen, bevor eine manuelle Fehlerbeseitigung eingeleitet werden kann. Für den Notfall und zum Schutz vor unbefugtem Zutritt zur Prüfwelle verfügt der Prüfautomat an verschiedenen Stellen über Notaus-Schalter, deren Zustände von der Safety SPS überwacht werden.

Die Prüfwelle ist aus einem mit acht Prüfteil-Aufnahmen versehenen Rundschalttisch und

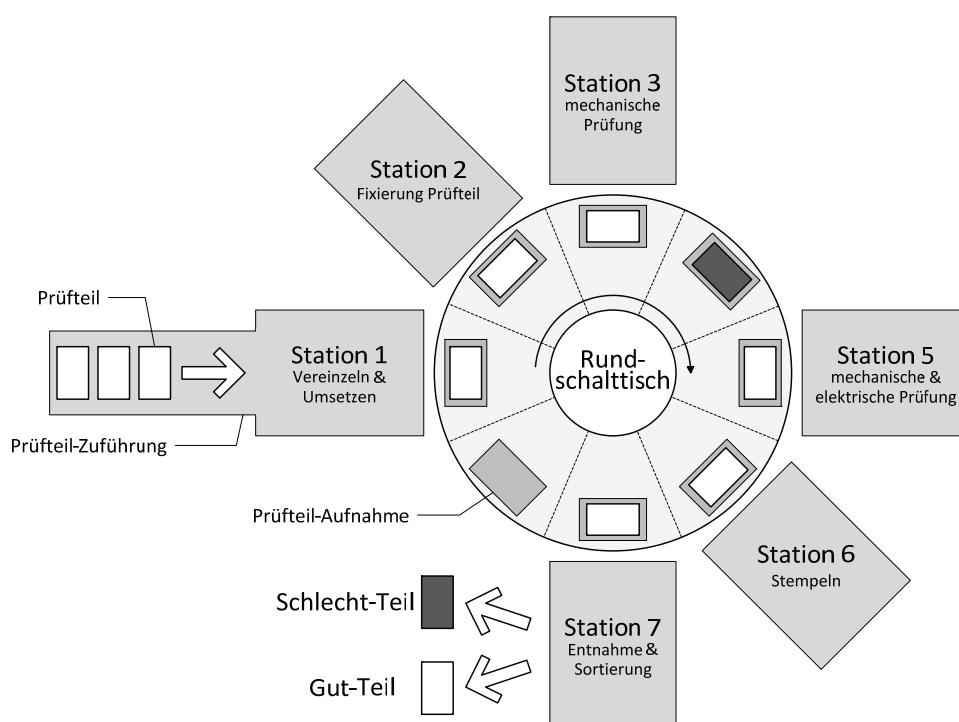


Abbildung 10-1: schematische Darstellung der Prüfwelle

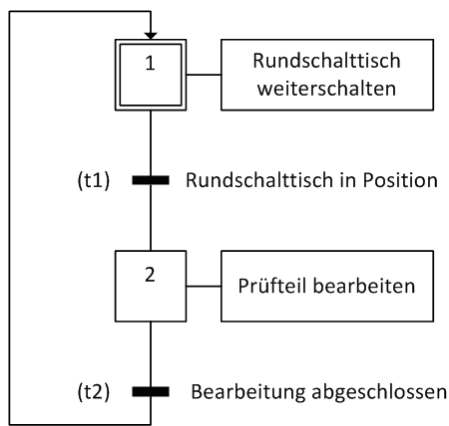


Abbildung 10-2: taktgetriebene Funktionsprüfung der Prüfstation

sechs angegliederten Prüfstationen aufgebaut (→ Abbildung 10-1). Alle vorhandenen Aktoren in den Prüfstationen werden pneumatisch (über pneumatische Linearantriebe), das Förderband und der Rundschalttisch jeweils durch einen Elektromotor angetrieben. Die Erkennung von Prüfteilen erfolgt über Lichtschranken bzw. Lichttaster ausschließlich an den Stationen 1 und 7. Die Positionen der Linearantriebe können in den jeweiligen Endlagen jeweils durch einen Näherungsschalter erfasst werden. Insgesamt sind 61 digitale Eingänge und 47 digitale Ausgänge vorhanden, welche an die SPS gekoppelt sind.

Die Prüfstation erlaubt die schrittweise Durchführung der Funktionsprüfung nach dem *first in – first out* Prinzip. Im stationären Betrieb sollen so mehrere Prüfteile gleichzeitig bearbeitet werden können. Durch die radiale Anordnung der Prüfteil-Aufnahmen lässt sich der Rundschalttisch in acht Sektoren einteilen, von denen sechs mit Prüfstationen belegt sind. Die schrittweise Funktionsprüfung innerhalb der Prüfstation selbst erfolgt taktgetrieben entlang der verschiedenen Bearbeitungsstationen 1, 2, 3, 5, 6 und 7. Dies bedeutet, dass während der Durchführung der Bearbeitungsvorgänge an den Stationen der Rundschalttisch in seiner aktuellen Position verharrt. Sind alle Bearbeitungsvorgänge beendet, so wird die Position des Rundschalttisches um einen Sektor im Uhrzeigersinn weiterschaltet. Somit ergibt sich für den Steuerungsablauf der Prüfstation ein taktgetriebenes Verhalten, welches durch das abwechselnde Weiterschalten des Rundschalttisches und der Bearbeitung der Prüfteile in den Bearbeitungsstationen geprägt ist (→ Abbildung 10-2). Die Station mit der längsten Bearbeitungszeit und die Dauer des Weiterschaltens bestimmen somit die Taktzeit der Prüfstation. Bezogen auf ein Prüfteil werden insgesamt sieben Takte für einen kompletten Durchlauf, also Funktionsprüfung und Sortierung in Gutteil oder Schlechteil, benötigt. Die Zuführung von Prüfteilen erfolgt über ein Förderband zur Station 1, welches während des Betriebs des Prüfautomaten kontinuierlich angetrieben wird.

Die einzelnen Bearbeitungsschritte innerhalb der Stationen 1, 2, 3, 5, 6 und 7 sollen nachfolgend beschrieben werden:

Station 1 (Vereinzeln & Umsetzen)

Die zugeführten Prüfteile werden in der Station 1 durch zwei Schieber vereinzelt und mit Hilfe eines Umsetzers vom Ende des Förderbands in die gegenüberliegende Prüfteil-Aufnahme des Rundschalttisches umgesetzt. Eine Vereinzelnung ist notwendig, da die Prüfteile ohne Abgrenzung auf dem Förderband positioniert sind. In der Ausgangssituation ist der erste Schieber in oberer, der zweite in unterer Position. Wird ein Prüfteil am Eingang von Station 1 erkannt, so fährt der erste Schieber in die untere und anschließend der zweite in die obere Endlage. Dadurch werden ein Prüfteil separiert und alle restlichen Prüfteile vor Station 1 aufgestaut. Das separierte Prüfteil fährt nun weiter zur Übergabestation, wo es durch eine Lichtschranke erfasst wird. Befindet sich das Prüfteil an der Übergabestation, so wird es durch den Umsetzer in die gegenüberliegende Prüfteil-Aufnahme verbracht. Hierzu fährt der Umsetzer mit geöffnetem Greifer in die untere Endlage, schließt die Greifzange, fährt in die

obere Endlage, fährt von der Übergabestation zur Prüfteil-Aufnahme, fährt herab und öffnet die Greifzange, um das Prüfteil abzusetzen. Anschließend fährt der Umsetzer zurück in seine Ausgangsposition.

Station 2 (Fixierung Prüfteil)

An dieser Station wird überprüft, ob das Prüfteil beim Umsetzen an Station 1 ordnungsgemäß auf der Prüfteil-Aufnahme fixiert ist. Ein vertikal beweglicher Schlitten mit zwei Dornen fährt dazu in die untere Endlage und verweilt dort einige Sekunden. Anschließend fährt der Schlitten zurück in die obere Endlage. Wurde beim Herunterfahren die untere Endlage nicht erreicht, so wird ein Fehlversuch registriert und der Vorgang ein weiteres Mal durchgeführt. Wurde dreimal in Folge ein Fehlversuch registriert, so wird eine quittierungspflichtige Störungsmeldung an den Anlagenbediener ausgegeben und die Anlage gestoppt, um Schäden an den nachfolgenden Stationen zu vermeiden.

Station 3 (mechanische Prüfung)

Station 3 dient der mechanischen Prüfung des Verschlusses. In einem ersten Schritt wird das Prüfteil vollständig fixiert. Dies erfolgt durch einen Niederhalter, der in seiner unteren Endlage ein Verrutschen des Prüfteils verhindert. Befindet sich der Niederhalter in der unteren Endlage, so fährt ein Schlitten über das Prüfteil. Anschließend wird der Verschluss durch das Ausfahren eines Dorns gespannt. Ist der Dorn wieder in seiner Ausgangslage angekommen, so fährt der Schlitten in seine Ausgangsposition zurück, wodurch der Verschluss mechanisch entspannt wird. Die mechanische Verschlussprüfung wird für jedes Prüfstück insgesamt fünfmal durchgeführt.

Station 5 (mechanische und elektrische Prüfung)

Die mechanische Prüfung an der Station 5 erfolgt analog zu Station 3. Zusätzlich erfolgt, parallel zur mechanischen Prüfung des Verschlusses, die elektrische Prüfung der Mikroschalter. Hierzu fährt ein horizontal beweglicher Prüfschlitten in Richtung Prüfteil. Die auf dem Prüfschlitten montierten Federkontakte erlauben eine elektrische Durchgangsmessung der Mikroschalter. Anschließend fährt der Prüfschlitten wieder in seine Ausgangsposition zurück und der Prüfvorgang beginnt erneut. Insgesamt werden drei Prüfungen pro Prüfteil durchgeführt. Werden alle mechanischen und elektrischen Prüfvorgänge an Station 5 erfolgreich absolviert, so wird das Prüfteil als Gutteil deklariert. Bereits bei einem erfolglosen Prüfvorgang erfolgt die Einstufung als Schlechteil.

Station 6 (Stempeln)

Handelt es sich um ein Gutteil, so wird das Prüfteil an der Station 6 mit einer vierstelligen Prüfnummer versehen. Dazu fährt ein horizontal beweglicher Stempelschlitten aus seiner Ausgangslage in Richtung Prüfteil. In seiner Endlage angekommen, wird der Stempel auf das Prüfteil aufgeprägt. Anschließend fährt der Schlitten in seine Ausgangslage zurück. Das Stempeln von Schlechteilen ist nicht vorgesehen, da sie an der Station 7 ausgesondert werden.

Station 7 (Entnahme & Sortierung)

An der Station 7 wird durch einen Lichttaster erfasst, ob sich ein Prüfteil in der Prüfteil-Aufnahme befindet oder nicht. In letzterem Fall wird der Bearbeitungsvorgang im aktuellen Takt nicht ausgeführt. Befindet sich hingegen ein Prüfteil in der Prüfteil-Aufnahme, so startet der Bearbeitungsvorgang. Hierzu fährt der Umsetzer von seiner horizontalen Ausgangs-

position in die vordere Endlage über das Prüfstück. Anschließend fährt der Umsetzer in die untere Endlage, schließt den Greifer, fährt in die obere Endlage und dann in Richtung der horizontalen Ausgangsposition. Handelt es sich bei dem aufgegriffenen Prüfteil um ein Gutteil, so fährt der Umsetzer direkt zur horizontalen Ausgangsposition zurück und setzt das Prüfstück an der Übergabeposition ab. Schlechttteile werden bei Erreichen der Schlechttteilposition zur Aussonderung abgeworfen.

Die beschriebenen Anforderungen an die Automatisierung des Prüfautomaten zeigen eine zunächst funktionsorientierte Sichtweise auf das Sollverhalten der Anlage. Bezogen auf den Steuerungsentwurf im Engineering automatisierter Anlagen ist diese Sichtweise charakteristisch für die Planungsphase, in deren Verlauf beispielsweise das Lastenheft entsteht. Der mit dem Lastenheft konfrontierte Steuerungsprogrammierer steht in nachfolgenden Engineering-Phasen vor der Aufgabe, die Anforderungen zu analysieren und aus dem beschriebenen Sollablauf des Prüfautomaten einen entsprechenden Steuerungsablauf abzuleiten und in einem Steuerungsprogramm zu implementieren. Neben dieser informellen Spezifikation steht dem Steuerungsprogrammierer dazu in der Regel eine Übersicht über die in der Anlage verbauten Sensoren und Aktoren und deren Parameter zur Verfügung (*Signalliste* → Anhang E).

Zur Lösung der im vorangegangenen Abschnitt in Textform (und nicht formal) spezifizierten Automatisierungsaufgabe ergeben sich folgende wesentliche Herausforderungen für den Steuerungsprogrammierer:

- Durch die Implementierung müssen einerseits die Bearbeitungsvorgänge innerhalb der Stationen automatisiert und andererseits der übergeordnete, getaktete Ablauf des Rundschalttischs und der Bearbeitungsstationen mit den vorhandenen Sensoren und Aktoren koordiniert werden.
- Es müssen Programme für die Betriebsarten *Automatikbetrieb* und *Handbetrieb* erstellt werden, wobei der Wechsel zwischen den Betriebsarten jederzeit möglich sein muss. Insbesondere in der Betriebsart *Handbetrieb* müssen mögliche Fehlerzustände des Prüfautomaten im Steuerungsprogramm berücksichtigt werden. Zusätzlich muss ein Notaus-Konzept implementiert werden, um die funktionale Sicherheit des Prüfautomaten sicherzustellen.
- Neben der *stationären Betriebsphase* des Prüfautomaten, in dem alle Prüfteil-Aufnahmen von Station 1 bis Station 7 in jedem Takt mit Prüfteilen belegt sind (→ Abbildung 10-1), müssen die Betriebsphasen *Anfahren* und *Herunterfahren* programmiertechnisch umgesetzt werden, in denen der Rundschalttisch nur teilweise belegt ist.

Aufgrund der heute üblichen, informellen Vorgehensweise einer direkten Implementierung werden die Anforderungen aus dem Lastenheft unmittelbar in ein Steuerungsprogramm überführt, mitsamt den bereits beschriebenen Auswirkungen (→ 2.1). Das bestehende Steuerungsprogramm des Prüfautomaten ist beispielsweise in der IEC 61131-3 Programmiersprache AWL implementiert und auf insgesamt 78 POEs, davon überwiegend Funktionsbausteine, verteilt. Hinsichtlich der Dokumentation der Automatisierungslösung erschwert diese

spezifische Sichtweise des Steuerungsprogrammierers auf den Steuerungsablauf den Bezug zum von der Planung geforderten Verhalten des Prüfautomaten.

10.2 Spezifikation des Steuerungsablaufs mit GRAFCET

Würden im betrachteten Fall des Prüfautomaten die funktionalen Anforderungen an die Automatisierung von Seiten der Planung in GRAFCET beschrieben, so ließe sich durch die im Rahmen dieser Arbeit vorgeschlagene formale Methodik automatisiert IEC 61131-3 Steuerungscode generieren. Dies würde hinsichtlich des Engineerings eine effizientere Zusammenarbeit zwischen Planung und Realisierung ermöglichen. Eine diesem *Brückenschlag* entsprechende, systematische Vorgehensweise auf der Grundlage formaler Methoden sowie darauf abgestimmte, prototypisch implementierte Engineering-Werkzeuge wurden im Rahmen dieser Arbeit vorgeschlagen (→ 6 - 9). Am Beispiel des Prüfautomaten soll nun aufgezeigt werden, wie die funktionalen Anforderungen des Lastenheftes in einem Grafcet formal beschrieben werden können (→ 10.2) und wie mit Hilfe der vorgeschlagenen Engineering-Werkzeuge automatisiert IEC 61131-3 Steuerungscode generiert werden kann (→ 10.3).

Im Zuge der Automatisierung des Prüfautomaten müssen unterschiedliche Aspekte berücksichtigt werden, wie beispielsweise die Betriebsarten *Automatikbetrieb* und *Handbetrieb* oder Bedienerückmeldungen und unterschiedliche Betriebsphasen. In Äquivalenz zur informellen Spezifikation müssen diese Aspekte und die jeweiligen Wirkzusammenhänge entsprechend des Planungsfortschritts in die Erstellung eines Grafcet einfließen. Somit spezifiziert der zu erstellende Grafcet die jeweiligen Steuerungsabläufe der einzelnen Bearbeitungsstationen auf der untersten Hierarchieebene, den übergeordneten getakteten Ablauf zwischen der Bearbeitung der Prüfteile und der Bewegung des Rundschalttisches (→ Abbildung 10-2) auf der mittleren Hierarchieebene sowie den Globalen Grafcet der Betriebsarten auf der obersten Hierarchieebene (→ Anhang F). Für die Erstellung kann sowohl eine *top-down* als auch eine *bottom-up* Vorgehensweise gewählt werden. Für die Verknüpfung der Teil-Grafcets mit den jeweils übergeordneten (Teil-) Grafcets stehen gemäß DIN EN 60848 zwei hierarchische Konstrukte zur Verfügung, *einschließende Schritte* (→ 4.3.2) und *zwangsstuernde Befehle* (→ 4.3.3). *Einschließende Schritte* ermöglichen eine eingängige grafische Darstellung der hierarchischen Verknüpfung zu den untergeordneten Teil-Grafcets der Bearbeitungsstationen durch jeweils einen *einschließenden Schritt*. Aus der Perspektive des Teil-Grafcets des Rundschalttisches lassen sich zudem die parallelen Bearbeitungsvorgänge in den Stationen grafisch kompakt hervorheben (G0, → Anhang F). Bei *zwangsstuernden Befehlen* wären für die hierarchische Einbindung der Bearbeitungsstationen jeweils zwei *zwangsstuernde Befehle* im Teil-Grafcet des Rundschalttisches notwendig, um gleiches dynamisches Verhalten zu spezifizieren. Die parallelen Bearbeitungsvorgänge in den Bearbeitungsstationen müssten durch eine umfangreichere Struktur grafisch hervorgehoben werden. Daher wird die hierarchische Struktur des Grafcet des Prüfautomaten mit *einschließenden Schritten* modelliert. Grundsätzlich sollte darauf geachtet werden, dass gleiche hierarchische Abhängigkeiten durch gleiche Konstrukte modelliert werden. Dies trägt wesentlich zur Übersicht des Grafcet bei.

Aus Gründen der Übersichtlichkeit werden die nachfolgend aufgeführten Einschränkungen für das Anwendungsbeispiel des Prüfautomaten festgelegt:

- Es wird ausschließlich die *stationäre Betriebsphase* betrachtet.
- Bedierrückmeldungen zur Anzeige am Text-Display, wie beispielsweise Störungsmeldungen, werden nicht explizit spezifiziert.
- Das für die Implementierung vorgesehene Notaus-Konzept wird in dem Grafcet nur durch ein *NOTAUS*-Signal angedeutet.

Für den Prüfautomaten wurde die Erstellung des Grafcet in der *top-down* Vorgehensweise durchgeführt. Die hierarchischen Abhängigkeiten wurden mit einschließenden Schritten modelliert. Abbildung 10-3 zeigt hierzu den Globalen Grafcet der Betriebsarten, welcher den Steuerungsablauf des Rundschalttisches in den Betriebsarten *Automatikbetrieb* und *Handbetrieb* spezifiziert. Durch den *einschließenden Schritt* s_3 wird der Teil-Grafcet des Rundschalttisches G0 in den übergeordneten Steuerungsablauf integriert. G0 wiederum enthält *einschließende Schritte*, welche die Teil-Grafcets der einzelnen Stationen in den Steuerungsablauf einbinden (\rightarrow Anhang F). Nach Initialisierung des Grafcet im Schritt s_1 kann zunächst zwischen den beiden Betriebsarten *Automatikbetrieb* und *Handbetrieb* gewählt werden. Ist der *einschließende Schritt* s_4 aktiviert, so wird dadurch gleichzeitig der *Handbetrieb* aktiviert. Ist der *einschließende Schritt* s_3 aktiviert, so erfolgt gleichzeitig die Aktivierung von s_{10} in G0, und der *Automatikbetrieb* startet. Der *Automatikbetrieb* soll nachfolgend in seinem grundsätzlichen Ablauf näher beschrieben werden (\rightarrow Anhang F):

Wenn der Rundschalttisch zu Beginn des *Automatikbetriebs* das Weiterschalten beendet hat ($t_{10} = true$), so werden die einschließenden Schritte $s_{11}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}$ zeitgleich aktiviert und die Bearbeitungsvorgänge in den Stationen unabhängig voneinander gestartet. Ist der Bearbeitungsvorgang in einer Station beendet, so löst die nachfolgende Transition aus, beispielsweise t_{11} für Station 1, und der nachfolgende Warteschritt wird aktiviert, beispielsweise s_{17} für Station 1. Aufgrund der Definition *einschließender Schritte* in [DIN EN 60848][#] ist sichergestellt, dass gleichzeitig mit der Deaktivierung des *einschließenden Schritts* alle *eingeschlossenen Schritte* in den untergeordneten Teil-Grafcets deaktiviert werden. Erst wenn alle Bearbeitungsvorgänge an den Stationen beendet sind, schaltet der Rundschalttisch weiter und ein neuer Zyklus beginnt. Aufgrund der pneumatischen Linearantriebe werden in dem Grafcet des Prüfautomaten als Aktionen überwiegend *gespeichert wirkende Aktionen*

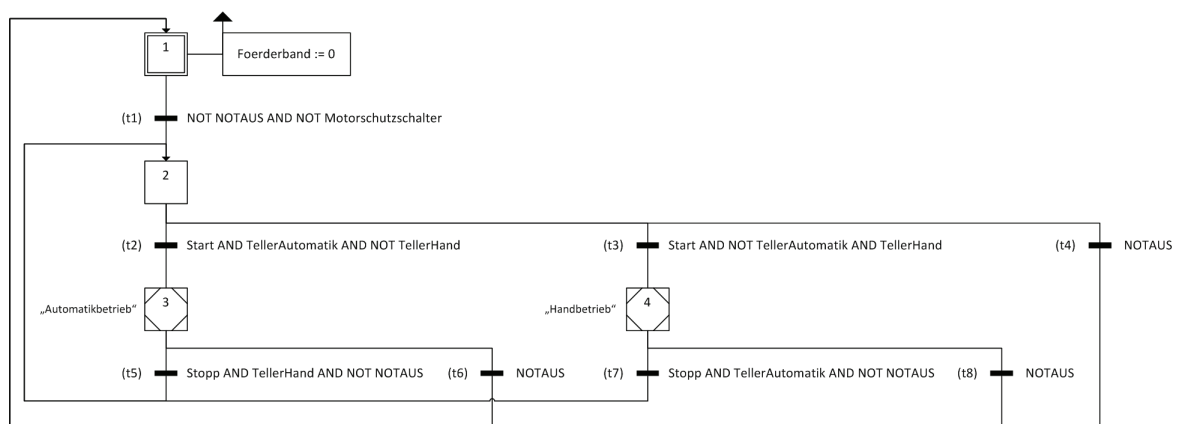


Abbildung 10-3: Teil-Grafcet des Rundschalttisches

verwendet, um die Steuerungsabläufe grafisch übersichtlich darzustellen. Würden stattdessen *kontinuierlich wirkende Aktionen* verwendet, deren Verwendung ebenfalls denkbar ist, so resultierten daraus zahlreiche Parallelverzweigungen innerhalb der einzelnen Teil-Grafcets. Während des Automatikbetriebs ist ein direktes Umschalten in den Handbetrieb nur gleichzeitig mit der Betätigung des *Stopp*-Tasters (→ Anhang E) möglich. Der Handbetrieb muss dann noch manuell durch Bestätigen des *Start*-Tasters gestartet werden. Aus Gründen der Übersichtlichkeit wird im Folgenden ausschließlich die Betriebsart *Automatikbetrieb* weiter betrachtet. Die Vorgehensweise zur Spezifikation der Betriebsart *Handbetrieb* kann analog erfolgen.

10.3 Transformation in IEC 61131-3 Steuerungscode

Die im vorangegangenen Abschnitt beschriebene methodische Vorgehensweise für den Steuerungsentwurf mit GRAFCET wird durch die prototypisch implementierten Engineering-Werkzeuge GRAFCET-Editor (→ 9.1) und GRAFCET-Transformator (→ 9.2) unterstützt. So kann der Grafcet des Prüfautomaten unmittelbar im GRAFCET-Editor erstellt und in einem nachfolgenden Arbeitsschritt automatisch in IEC 61131-3 konformen Steuerungscode überführt werden (→ Abbildung 10-4). Im GRAFCET-Editor wird zunächst der Globale Grafcet der Betriebsarten des Prüfautomaten entworfen. Ausgehend von den *einschließenden Schritten* können dann durch die Auswahl eines Menübefehls die einzelnen Teil-Grafcets des Rundschalttisches und der Bearbeitungsstationen (→ Anhang F) jeweils in einem gesonderten Zeichenblatt entworfen werden. Eine entsprechende Verknüpfung zwischen *einschließendem Schritt* und zugehörigem Teil-Grafcet der eingeschlossenen Schritte wird im Datenmodell des GRAFCET-Editors hinterlegt.

Zur vereinfachten Spezifikation von Transitionsbedingungen und Aktionen verfügt der GRAFCET-Editor über eine nutzerspezifisch anpassbare Datenschnittstelle zu *MS Excel*, so dass die in einer Signalliste bereits vorliegenden Signale (→ Anhang E) auch bei der Erstellung des Grafcet genutzt werden können. Durch manuell auswählbare Prüfroutinen zur Konsistenzprüfung wird die Syntax des Grafcet hinsichtlich der in IEC 60848 festgelegten

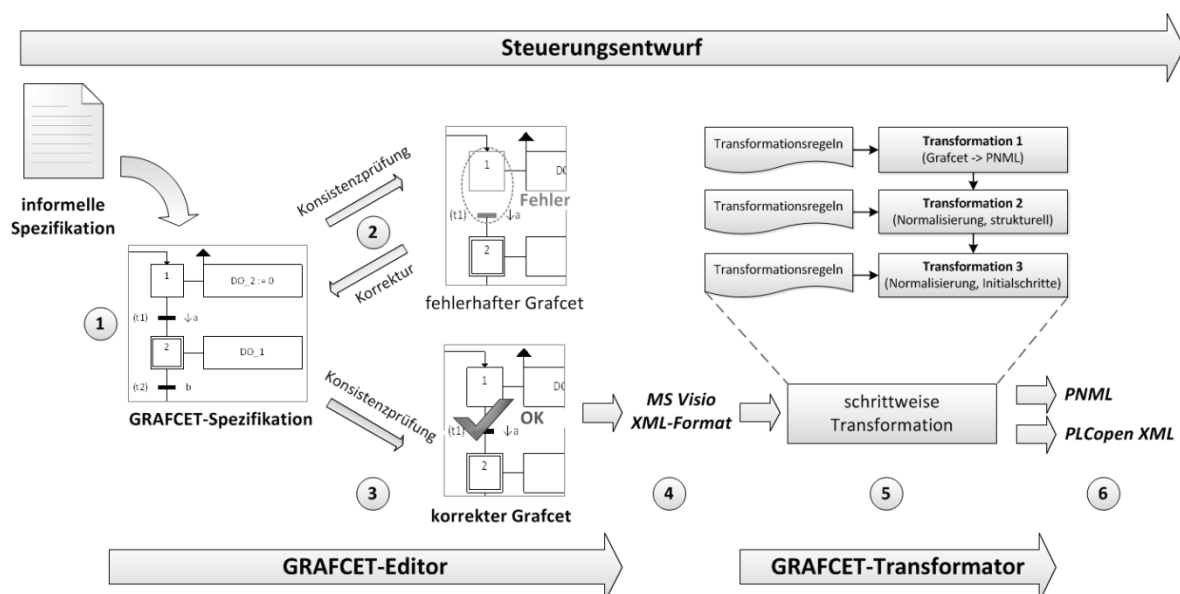


Abbildung 10-4: Arbeitsweise mit GRAFCET-Editor und -Transformator

Grundsätze analysiert. Für eine weitere Bearbeitung durch den GRAFCET-Transformator sollten ausschließlich erfolgreich geprüfte Grafquets vorgesehen werden, da im Rahmen der Konsistenzprüfung wesentliche Informationen über die Verknüpfungen der einzelnen GRAFCET-Elemente im Datenmodell des GRAFCET-Editors hinterlegt werden.

Für das Beispiel des Prüfautomaten wurden der Teil-Grafquet des Rundschalttisches sowie die Teil-Grafquets der Bearbeitungsstationen für die Betriebsart *Automatikbetrieb* im GRAFCET-Editor modelliert. Der zugehörige Grafquet besteht aus insgesamt 60 Schritten, 62 Transitionen, 46 *gespeichert wirkenden Aktionen*, 15 *kontinuierlich wirkenden Aktionen* und 6 *einschließenden Schritten*, die auf die Teil-Grafquets der Bearbeitungsstationen verweisen. Die Datei wird im *MS Visio* spezifischen XML-Datenformat abgespeichert und bildet das Eingangsdatenformat des GRAFCET-Transformators. Im GRAFCET-Transformator wählt der Anwender die *MS Visio*-XML-Datei und das Zielverzeichnis aus, in dem er die Ausgabedatei des GRAFCET-Transformators ablegen möchte. Daraufhin wird die Transformation gestartet. Als Dateiformate für die Ausgabedatei stehen PNML, PNML in normalisierter Form (NPNML) und *PLCopenXML* (→ 9.2.1) zur Verfügung. Für das Anwendungsbeispiel des Prüfautomaten soll das Hauptaugenmerk auf dem *PLCopenXML*-Format liegen, um die einzelnen Arbeitsschritte von der informellen Spezifikation zum Steuerungscode zu veranschaulichen. Wurde die Transformation und automatische Generierung des Steuerungscode erfolgreich abgeschlossen, so wird dies in einem Anzeigefeld des GRAFCET-Transformators angezeigt. Die *PLCopenXML*-konforme Ausgabedatei befindet sich im angegebenen Zielverzeichnis und steht für einen Import in IEC 61131-3 Programmierwerkzeuge zur Verfügung (→ 3.6). Für die durch den GRAFCET-Transformator automatisch generierte *PLCopenXML*-Datei lassen sich die folgenden Ergebnisse festhalten:

Die im Deklarationsteil aufgeführten Variablen, Transitionsbedingungen und Aktionen werden in Übereinstimmung mit den Transformationsregeln in den Steuerungscode übertragen. Darüber hinaus entspricht die Struktur (Anweisungsteil) des automatisch generierten SFCs des Prüfautomaten (→ Anhang G) dem erwarteten Transformationsergebnis, welches sich durch manuelle Anwendung der Normalisierungsschritte nachvollziehen lässt. Die Struktur des automatisch generierten SFCs besteht aus insgesamt 61 Schritten und 109 Transitionen, wobei die Schritte und Transitionen des Grafquets des Prüfautomaten beibehalten werden. Die Abweichungen zum Grafquet des Prüfautomaten resultieren einerseits aus dem zusätzlichen Initialschritt des SFC und andererseits aus den zusätzlichen Wirkverbindungen der *einschließenden Schritte* zu den untergeordneten Teil-Grafquets der eingeschlossenen Schritte. Diese *impliziten Wirkverbindungen* zwischen dem Teil-Grafquet des Rundschalttisches und den Teil-Grafquets der Bearbeitungsstationen werden durch die Normalisierung zu expliziten Wirkverbindungen. Sie erfordern jeweils eine zusätzliche Transition, die von dem *eingeschlossenen Schritt* zu den Schritten im Nachbereich des *einschließenden Schritts* verläuft. Für den Grafquet des Prüfautomaten besteht der Nachbereich der *einschließenden Schritte* ($S_{11}, S_{12}, S_{13}, S_{14}, S_{15}, S_{16}$) jeweils aus nur einem Schritt (z. B.: S_{17} im Falle von S_{11}).

In einem weiterführenden Schritt wurde ein Ausschnitt des Grafquets des Prüfautomaten betrachtet und gemäß der Normalisierungsschritte und Transformationsregeln manuell als SFC in *Multiprog* implementiert (→ Abbildung 10-5). *Multiprog* unterstützt den *PLCopenXML*-Export für grafische SFCs (→ 3.6), so dass ein Vergleich des manuell implementierten SFCs

in grafischer Form (*Multiprog*) und des automatisch generierten SFCs (GRAF CET-Transformator) auf Basis der *PLCopenXML*-Dateien möglich ist. Der Ausschnitt des Graficet umfasst ausschließlich den Teil-Graficet des Rundschalttisches sowie den untergeordneten Teil-Graficet der Bearbeitungsstation 2 und enthält 8 Schritte, 10 Transitionen, 4 *kontinuierlich wirkende Aktionen* und 3 *gespeichert wirkende Aktionen*.

Insgesamt betrachtet sind sowohl die *Multiprog*-Exportdatei als auch die automatisch generierte Datei des GRAFCET-Transformators konform zu den Definitionen in [PLCOPEN 2009][#]. Die Analyse zeigte zudem, dass die Deklarationen von Variablen, Aktionen und Transitionsbedingungen in beiden *PLCopenXML*-Dateien in gleicher Art und Weise abgelegt werden. Schritte und Transitionen werden im Anweisungsteil der *Multiprog*-Exportdatei als XML-Elemente abgelegt (→ Abbildung 10-5). Als weitere grafische Elemente eines SFC sind Aktionen in Form von Aktionsblöcken, Parallelverzweigungen und Sprunganweisungen explizit im Anweisungsteil der *Multiprog*-Exportdatei enthalten. Wirkverbindungen werden für grafische SFCs durch die Referenzierung von Identifikationsnummern der jeweiligen Vorgängerelemente in *PLCopenXML* abgelegt. Im Fall von Schritten und Parallelverzweigungen wird auf die Vorgängertransitionen, im Fall von Transitionen auf die Vorgängerschritte und vorausgehenden Parallelverzweigungen verwiesen. Weitere Verbindungen, wie beispielsweise die Verbindung eines Aktionsblocks mit einem Schritt oder die direkte Verknüpfung einer Eingangsvariablen mit einer Transition,

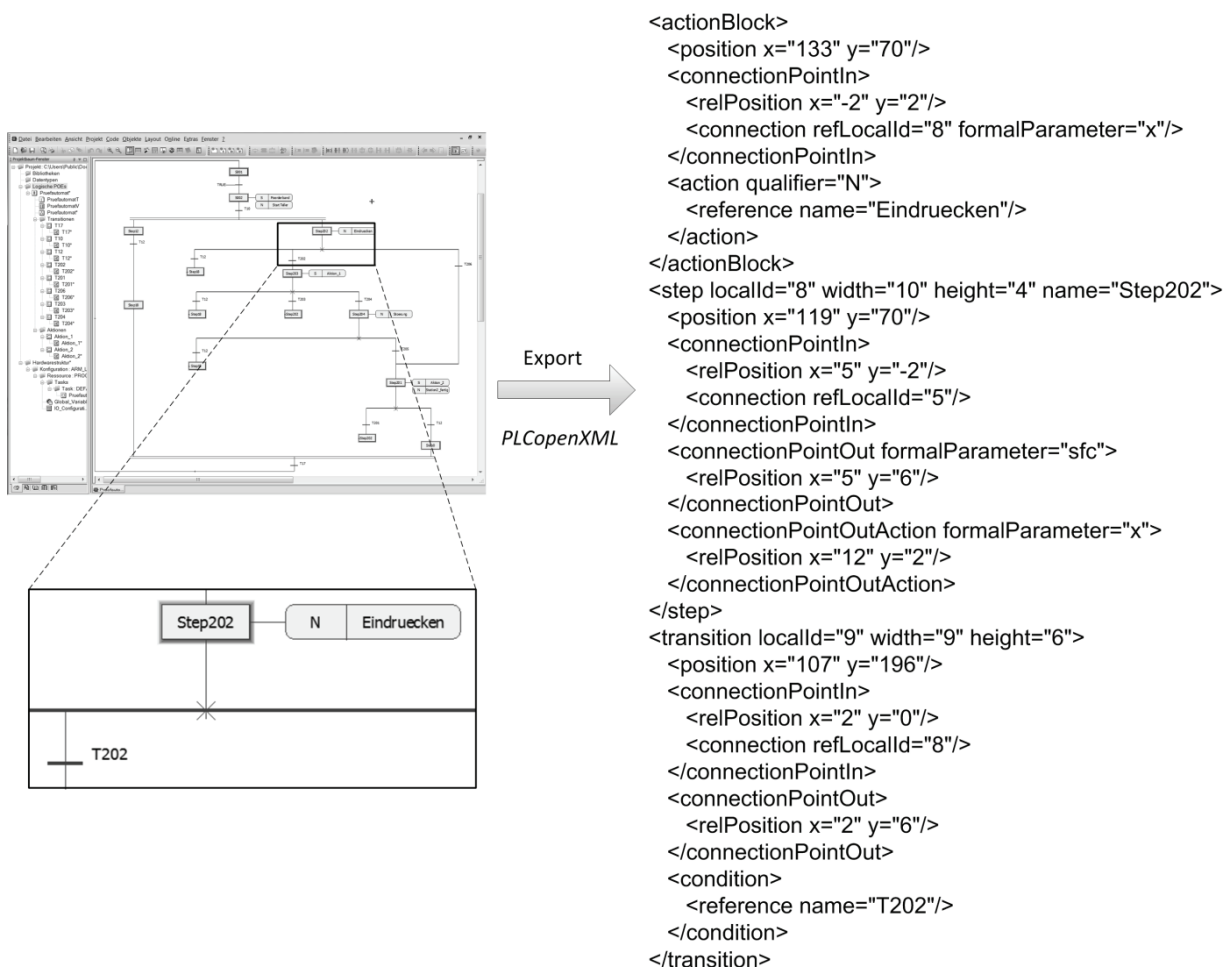


Abbildung 10-5: Manuell erstellter SFC in *Multiprog* und Auszug aus der zugehörigen *PLCopenXML*-Exportdatei

können durch die notwendigen Positionskoordinaten berechnet werden (→ 8.2.2). Bezüglich der Aktionen in *Multiprog* ist festzuhalten, dass für einfache Transformationen (Transformationsregeln 11, 14, 15, 21, 22, → Anhang D) keine zusätzlichen Aktionen im Deklarationsteil notwendig sind, sondern die Aktionen direkt mit den Schritten assoziiert werden können. In der *Multiprog*-Exportdatei resultiert dies einerseits in einem reduzierten Deklarationsteil, erfordert aber andererseits das Speichern der Aktionen als Aktionsblöcke im Anweisungsteil (→ Abbildung 10-5).

Die automatisch generierte *PLCopenXML*-Datei des GRAFCET-Transformators hingegen enthält den SFC in Textform (→ Anhang G). Wirkverbindungen werden innerhalb der Transitionen so angegeben, dass jede Transition die Information enthält, mit welchen Vorgänger- und Nachfolgeschritten sie verbunden ist. Den Schlüsselwörtern *FROM* und *TO* folgt dazu die Angabe der Schritte im Vor- und Nachbereich der Transition. Parallelverzweigungen werden, im Gegensatz zur grafischen Repräsentation, nicht als gesonderte SFC-Elemente berücksichtigt. Sie sind indirekt in den Definitionen der Transitionen enthalten. Aktionen werden im Gegensatz zur *Multiprog*-Exportdatei einheitlich im Deklarationsteil definiert. In den Schrittdefinitionen des Anweisungsteils werden die entsprechenden Aktionen dann referenziert.

Am Beispiel von *CODESYS* konnte darüber hinaus aufgezeigt werden, dass der automatisch generierte Steuerungscode des Prüfautomaten über die *PLCopenXML*-Schnittstelle in das SPS-Programmierwerkzeug importiert werden kann. Der Deklarationsteil für Variablen, Transitionen und Aktionen sowie der Anweisungsteil wurden in einem Programm abgelegt und standen für die Bearbeitung in *CODESYS* zur Verfügung, wie beispielsweise die Ergänzung von Hardware-Adressen der Signale. Aufgrund der fehlenden Unterstützung einer IEC 61131-3 konformen textuellen Repräsentation von SFCs wurde der Anweisungsteil des Steuerungscode zwar als ST-Programm identifiziert, die Schlüsselwörter *STEP* und *TRANSITION* konnten von *CODESYS* allerdings nicht interpretiert werden. Somit wurde der importierte Steuerungscode nicht unmittelbar als SPS-Programm kompiliert.

11 Zusammenfassung und Ausblick

11.1 Zusammenfassung

Die heute übliche Vorgehensweise im Steuerungsentwurf automatisierter Anlagen ist geprägt durch einen dokumentenbasierten Austausch von Engineering-Informationen. Insbesondere die Schnittstelle zwischen Spezifikation und Implementierung, die oftmals einher geht mit der Gewerkegrenze zwischen Planung und Realisierung, ist davon betroffen, so dass wesentliche Zusammenhänge und Daten manuell ausgewertet und interpretiert werden müssen. Nicht selten werden durch diesen informellen Datenaustausch Engineering-Tätigkeiten doppelt ausgeführt, was zu erheblichem Zeitdruck in der Fertigstellung des Automatisierungssystems führt. Die heute übliche intuitive Vorgehensweise der direkten Implementierung führt, gemeinsam mit stetig steigenden zeitlichen und qualitativen Anforderungen in einem zunehmend komplexeren Automatisierungsumfeld, zu Fehlern in den Steuerungsprogrammen, deren Ursache in der Entwurfsphase liegt und deren Auswirkungen erst während des Anlagenbetriebs entdeckt werden. Die Beseitigung der entdeckten Fehler verursacht zum Teil erhebliche Mehrkosten. Zudem fehlt es an einer gewerkeübergreifenden, nachvollziehbaren und wartbaren Dokumentation des Steuerungsablaufs neben den Steuerungsprogrammen selbst. Marktübliche Werkzeuge bieten zwar umfassende Möglichkeiten zur Implementierung standardisierter und herstellerspezifischer SPS-Programme, vernachlässigen aber weitestgehend eine Integration geeigneter grafischer Beschreibungsmittel zur Unterstützung eines systematischen Steuerungsentwurfs im Sinne des Model Driven Engineerings. Das Steuerungsprogramm ist somit die einzige Quelle über die tatsächliche Implementierung des Steuerungsablaufs.

Grafische Beschreibungsmittel zur Spezifikation von Steuerungsabläufen im Zusammenhang mit formalen Methoden zeigen geeignete Ansatzpunkte für die Unterstützung eines systematischen Steuerungsentwurfs. Der automatischen Generierung von Steuerungscode kommt dabei eine Schlüsselrolle zu. Dennoch haben formale Methoden bisher wenig Akzeptanz in der praktischen Anwendung gefunden. Eine mögliche Ursache hierfür ist, dass Steuerungsprogrammierer aufgrund ihrer beruflichen Ausbildung zumeist keine Kenntnisse dieser semi-formalen bzw. formalen Beschreibungsmittel und Methoden, wie beispielsweise *UML-Aktivitätsdiagramme* oder *Net Condition / Event Systems*, besitzen, so dass die entsprechenden Forschungsansätze nur schwer in der Praxis integriert werden können. Die vorhandenen unzureichenden Kenntnisse können in diesem Fall nur durch zusätzliche Schulungen erweitert werden. Sie finden daher entsprechend wenig Anklang in den Unternehmen.

Die vorliegende Arbeit widmet sich aus diesem Grund dem Beschreibungsmittel GRAFCET und schlägt einen anderen Weg für die Einführung formaler Methoden im Steuerungsentwurf vor. GRAFCET ist ein durch Petrinetze inspiriertes grafisches Beschreibungsmittel zur Spezifikation von Steuerungsabläufen und seit dem Jahre 1988 internationaler Standard IEC 60848. Neben der Darstellung einfacher Schrittkettenstrukturen sowie alternativer und nebenläufiger Verzweigungen ermöglicht GRAFCET darüber hinaus eine hierarchische, modulare Strukturierung des Steuerungsablaufs und die Spezifikation zeitabhängiger Anforderungen. GRAFCET ersetzt seit dem Jahr 2002 den zuvor in Deutschland gültigen Standard DIN 40719-6 „Regeln für Funktionspläne“ und ist mittlerweile verpflichtender

Bestandteil von Zwischen- und Abschlussprüfungen in der beruflichen Ausbildung. Ein systematischer Steuerungsentwurf mit GRAFCET, zu dem diese Arbeit beiträgt, spricht folglich die Steuerungsprogrammierer direkt an und beinhaltet das Potential, in der praktischen Anwendung akzeptiert zu werden.

Allerdings ist GRAFCET lediglich textuell im Standard IEC 60848 definiert und somit der Klasse der semi-formalen Beschreibungsmittel zuzuordnen. Um formale Methoden, insbesondere Algorithmen zur automatischen Generierung von Steuerungscode, allgemein-gültig auf GRAFCET anwenden zu können, bedarf es zunächst einer geeigneten formalen Definition von Struktur und dynamischem Verhalten, die insbesondere *zeitabhängige Bedingungen* und die Elemente zur hierarchischen Strukturierung berücksichtigt. Darauf aufbauend muss seitens der Werkzeuge eine geeignete Infrastruktur verfügbar sein, welche die konzeptionellen Vorgaben eines systematischen Steuerungsentwurfs mit GRAFCET für den Anwender bereitstellt.

Hierzu wurde im Rahmen der vorliegenden Arbeit ein Konzept vorgestellt, mit welchem auf Basis eines Grafcet und unter Anwendung rechnergestützter, formaler Methoden Steuerungscode generiert werden kann, der den Anforderungen der IEC 61131-3 entspricht. Des Weiteren wurden softwarebasierte Werkzeuge prototypisch implementiert, die das Erstellen eines Grafcet sowie eine automatische Generierung ermöglichen und dadurch eine geeignete Infrastruktur für eine rechnergestützte Transformation zur Verfügung stellen. Bezüglich des Beschreibungsmittels GRAFCET, der angewendeten formalen Methoden und der unterstützenden Werkzeuge wurden im Rahmen dieser Arbeit Ergebnisse erzielt, die in den nachfolgenden Abschnitten zusammengefasst werden sollen.

Beschreibungsmittel GRAFCET

In einem ersten Schritt bestand die Notwendigkeit, Gemeinsamkeiten und Unterschiede von GRAFCET und SFCs näher zu beleuchten, um eine eindeutige Abgrenzung zwischen der Spezifikationsprache GRAFCET und der Programmiersprache SFC zu erreichen.

In einem weiteren Schritt wurde GRAFCET gemäß IEC 60848 bezüglich seiner Struktur und seines dynamischen Verhaltens formal als *steuerungstechnisch interpretiertes Petrinetz* beschrieben. Ausgehend von einer bereits existierenden formalen Definition der grundlegenden GRAFCET-Elemente (Basic-Grafcet), lag das Hauptaugenmerk auf der Erweiterung dieses formalen SIPN-Modells um *einschließende Schritte*, *zwangsstuernde Befehle* und *zeitabhängige Bedingungen*. In allen diesen Fällen wurde aufgezeigt, dass eine Rückführung der erweiterten Sprachkomponenten auf Elemente des Basic-Grafcet formal möglich ist. *Zeitabhängige Bedingungen* konnten außerdem formal auf *T-timed Petrinetze* zurückgeführt werden. Diese mit dem Begriff der Normalisierung umschriebene Rückführung wurde als wesentliche Voraussetzung für die Transformation hierarchisch strukturierter Grafkets in ein Steuerungsprogramm nach IEC 61131-3 identifiziert, da SFCs kein zu GRAFCET äquivalentes Hierarchiekonzept besitzen und zudem nur einen Initialschritt erlauben. Das entsprechende formale SIPN-Modell beinhaltet somit alle zentralen Elemente, die im Standard IEC 60848 für GRAFCET vorgesehen sind.

Um eine Integration in heutige Engineering-Werkzeuge sowie eine rechnergestützte Handhabung der Spezifikationsdaten zu gewährleisten, wurde darüber hinaus eine XML-basierte

implementierungsunabhängige Notation für das Beschreibungsmittel GRAFCET konzeptionell erarbeitet und mit Hilfe der *Petri Net Markup Language* umgesetzt. Somit können die GRAFCET-spezifischen Modellzusammenhänge in einem offenen XML-Datenaustauschformat dargestellt werden.

Methode

Der systematische Steuerungsentwurf mit GRAFCET wurde im Rahmen dieser Arbeit durch eine methodische Vorgehensweise zur automatischen Generierung IEC 61131-3 konformen Steuerungs-codes angereichert. Dabei wurden zwei Kernelemente der Transformation beschrieben, die Normalisierung und die Transformationsregeln. Als Vorstufe zur Umsetzung der automatischen Generierung stellt die Normalisierung zunächst sicher, dass die strukturellen Voraussetzungen für eine Transformation gegeben sind, so dass darauf folgend die definierten Transformationsregeln angewendet werden können.

Im weiteren Verlauf wurden Transformationsregeln auf Modellebene festgelegt, die eine Abbildung der im formalen Modell beschriebenen GRAFCET-Elemente in dazu äquivalente Programmkonstrukte der IEC 61131-3 erlauben. Für die Definition der Transformationsregeln wurde insbesondere von den aufgezeigten Gemeinsamkeiten zwischen SFCs und GRAFCET Gebrauch gemacht. Zusätzlich wurden Transformationsregeln auf Metamodellebene definiert, um eine rechnergestützte Transformation der implementierungsunabhängigen GRAFCET-Notation in das IEC 61131-3 konforme *PLCopenXML*-Datenaustauschformat umzusetzen. Die zugrunde gelegten Transformationsregeln sind grundsätzlich bidirektional gültig, wurden im Zusammenhang dieser Arbeit aber lediglich unidirektional von der Spezifikation zur Implementierung angewandt.

Werkzeugunterstützung

Aufgrund des beschriebenen Mangels an GRAFCET-Editoren, welche die Erstellung eines Graficet unterstützen und die hinterlegten Daten in einem rechnerlesbaren Datenformat abspeichern, wurde zunächst eine prototypische Applikation innerhalb der Büro-Anwendersoftware *Microsoft® Visio* implementiert, mit der Graficets in einer gewohnten Softwareumgebung erstellt werden können. Für das weiterführende Transformationswerkzeug wurde eine Importschnittstelle entwickelt, die *Microsoft® Visio*-spezifische XML-Dateien auslesen und nach GRAFCET-spezifischen Informationen durchsuchen kann. Ein zur Laufzeit der Transformation generiertes internes Objektmodell wurde als zentrale Datendrehscheibe implementiert und ermöglicht das Schreiben von Ausgabedateien im Format der *Petri Net Markup Language* oder dem *PLCopenXML*-Format.

11.3 Ausblick

Durch die vorliegende Arbeit wurden die wesentlichen Voraussetzungen für einen systematischen Steuerungsentwurf mit GRAFCET im Sinne des Model Driven Engineerings geschaffen. Es besteht nunmehr die Möglichkeit, GRAFCET zu verwenden, um Steuerungen zu spezifizieren und durch die Anwendung von Transformationsalgorithmen rechnergestützt IEC 61131-3 konformen Steuerungscode zu generieren.

Kurzfristig gesehen sollten anknüpfende Forschungsarbeiten das Hauptaugenmerk auf die im Rahmen dieser Arbeit entwickelten Werkzeugprototypen legen. So bieten *Microsoft*[®] *Visio* und *Visual Studio* zahlreiche Möglichkeiten, um die bisherigen Funktionen des GRAFCET-Editors und des GRAFCET-Transformators weiter auszubauen und bis zur Marktreife zu verfeinern. Hierfür sollten zunächst umfangreiche systematische Tests anhand von Referenz-Anlagen durchgeführt werden, um fehlerhafte Implementierungen hinsichtlich des GRAFCET-Transformators zu erkennen und notwendige Anpassungen durchzuführen. Daran anknüpfend sollte angestrebt werden, die aufgezeigten Transformationsregeln unabhängig von der Transformationsfunktionalität in einem dafür geeigneten Datenformat abzulegen, so dass zukünftige Ergänzungen oder Änderungen der IEC 60848 mit wenig Aufwand berücksichtigt werden können. Des Weiteren sollte ein Konzept entwickelt und umgesetzt werden, welches beschreibt, wie die Anforderungen potentieller Anwender systematisch erhoben werden und in die Softwareentwicklung einfließen können, so dass die methodische Vorgehensweise auch für reale Anwendungen Akzeptanz findet. Da momentan noch keine spezielle Anwendungsdomäne innerhalb des Engineerings automatisierter Anlagen feststeht, könnte diese durch entsprechende Umfragen und Analysen identifiziert werden. Folglich könnte eine zielgerichtete Strategie zur Integration des vorgestellten Ansatzes in die praktische Anwendung abgestimmt werden.

In Verbindung mit den Anwenderanforderungen scheint darüber hinaus mittelfristig eine tiefgreifende Analyse der für den systematischen Steuerungsentwurf notwendigen Engineering-Daten sinnvoll zu sein. So fokussiert die im Rahmen dieser Arbeit entwickelte Methode zur (teil-) automatischen Generierung nach derzeitigem Stand auf die logischen Zusammenhänge des Steuerungsablaufs. Das Beispiel der Import-Schnittstelle des GRAFCET-Editors für eine *Microsoft*[®] *Excel*-basierte, rudimentäre Signalliste zeigt allerdings grundsätzlich, dass vorhandene Engineering-Daten durch rechnergestützte Methoden zielführend in den Steuerungsentwurfsprozess integriert werden können. Inwiefern weitere vorhandene Anlagen- und Prozessdaten genutzt werden und aus welchen Modellen diese extrahiert werden können, wäre das Resultat einer entsprechenden Untersuchung. Grundsätzlich bestünde somit die Möglichkeit, nicht nur den Steuerungsablauf, sondern ein lauffähiges Steuerungsprogramm mitsamt der notwendigen Hardwarekonfigurierung automatisch zu generieren und in ein SPS-Programmierwerkzeug zu importieren.

Langfristig gesehen sollte die methodische Vorgehensweise zur automatischen Generierung IEC 61131-3 konformen Steuerungscode durch die automatische Generierung von Grafcets aus Steuerungsprogrammen ergänzt werden, um ein Re-Engineering zu ermöglichen. Somit könnten auch bereits bestehende Steuerungsprogramme formalen Methoden zugeführt werden, die mit dem formalen Modell für GRAFCET, welches auf *Steuerungstechnisch*

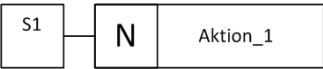
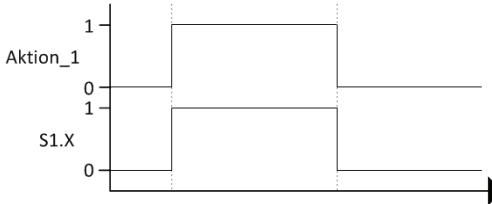
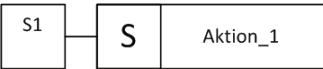
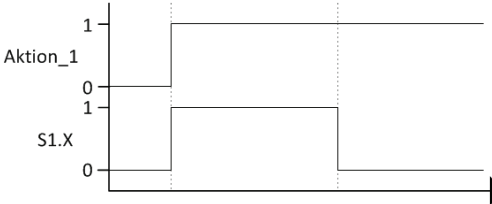
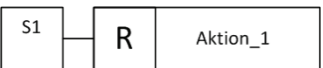
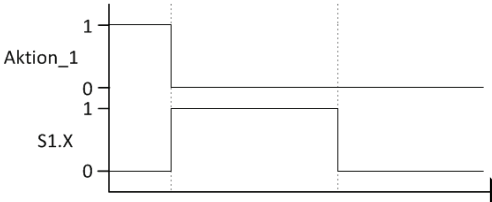
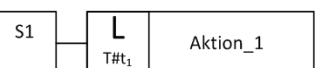
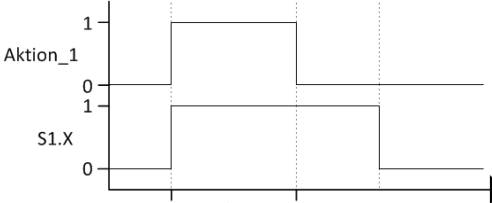
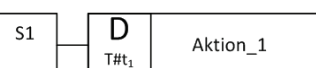
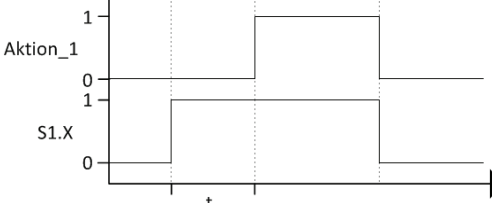
Interpretierten Petrinetzen basiert, verknüpft sind. Ansätze für ein solches Re-Engineering finden sich beispielsweise in [BANI YOUNIS 2006].

Die Quantifizierung der Vorzüge des vorgestellten, auf GRAFCET basierenden Konzepts zur automatischen Generierung IEC 61131-3 konformen Steuerungscode gegenüber der heute üblichen Praxis des Steuerungsentwurfs wird wohl nur schwer möglich sein. Eine Evaluation im Umfeld der zukünftigen potentiellen Anwender (z. B. Schüler in der beruflichen und berufsfachlichen Ausbildung) könnte diesbezüglich Aufschluss geben, erfordert allerdings hohen Aufwand hinsichtlich der Vorbereitung und Durchführung. Aber auch so zeigt die in aktuellen Anwenderumfragen dokumentierte Unzufriedenheit mit Engineering-Schnittstellen [HAPPACHER 2012], dass formale Methoden und insbesondere die automatische Generierung mehr denn je gefragt sind, um ein effizienteres Engineering in der praktischen Anwendung zu etablieren

Anhang A Historische Entwicklung der IEC 60848


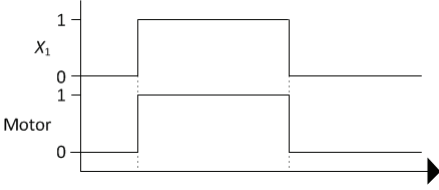
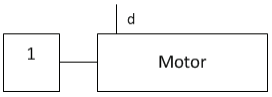
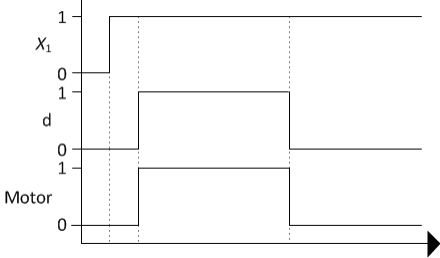
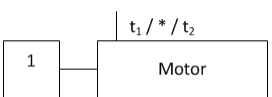
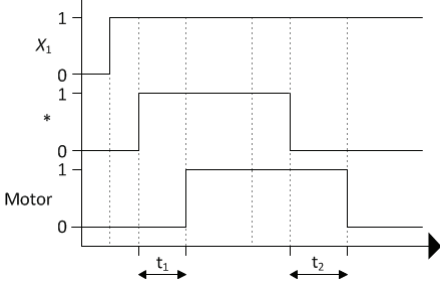
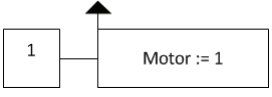
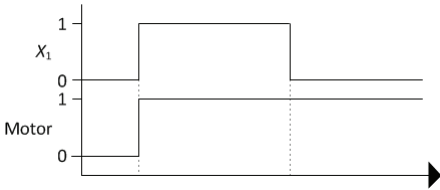
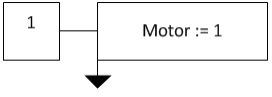
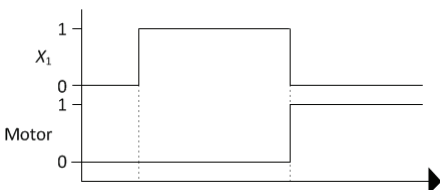
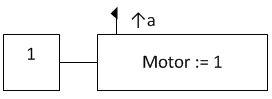
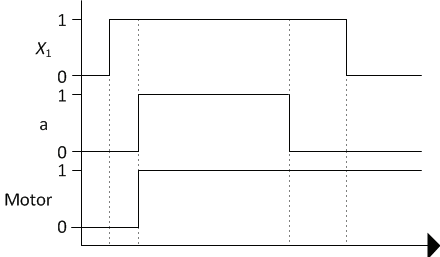
Jahr	Titel / Bezeichnung
1962	Die Dissertation von Carl Adam Petri mit dem Titel „Kommunikation mit Automaten“ wird veröffentlicht.
1969	Speicherprogrammierbare Steuerungen werden auf dem Markt eingeführt.
1970-1977	Die Grundzüge von GRAFCET werden durch die AFCET (Association Francaise pour la Cybernétique Economique et Technique) erarbeitet.
1977	Die ADEPA (Agence pour le Développement de la Productique Appliquée à l'Industrie) wird gegründet.
1978	GRAFCET wird in Frankreich Teil der Lehrpläne an technischen Gymnasien.
1982	GRAFCET wird französische Norm (NF C03-190).
1985	Statecharts werden von David Harel zur "Spezifikation reaktiver Systeme" beschrieben in [Harel et al. 85].
1986	P.J. Ramadge, W.M. Wonham veröffentlichen ihren Ansatz zur "Supervisory control theory" in [Ramadge et al. 86].
1987/88	GRAFCET wird erstmals internationale Norm (IEC 848 „Preparation of Function Charts for Control Systems“).
1992	Die erste überarbeitete Version der IEC 848 „Preparation of Function Charts for Control Systems“ wird veröffentlicht.
1992	Teil 1 und Teil 2 der IEC 1131 werden veröffentlicht.
1992	DIN 40719-6 „Schaltungsunterlagen-Regeln für Funktionspläne“ wird veröffentlicht.
1992/93	Die französische Union Technique d'Electricité (UTE) veröffentlicht ihre Publikation C 03-191 „Function Charts GRAFCET-Extensions of Basic Principles“.
1993	Teil 3 der IEC 1131 wird veröffentlicht.
1996	Durch eine Neunummerierung aller IEC-Normen werden die Normen für GRAFCET und SFC nun unter IEC 60848 (GRAFCET) und IEC 61131-3 (SFC) geführt.
1999	Im Rahmen der zweiten Überarbeitung der IEC 60848 wird das Arbeitspapier DIN IEC 3B/256/CD „Entwurfssprache GRAFCET für Ablauf-Funktionspläne“ veröffentlicht.
2000	Im Rahmen der zweiten Überarbeitung der IEC 60848 wird das Arbeitspapier IEC 3B/304/CDV „Specification Language GRAFCET for sequential function charts“ veröffentlicht.
2002	IEC 60848:2002 „GRAFCET-specification language for sequential function charts“ wird veröffentlicht und wenig später in DIN EN 60848 umgesetzt.
2003	IEC 61131-3:2003 „Programmable Controllers-Part 3: Programming Languages“ wird veröffentlicht und wenig später in DIN EN 61131-3 umgesetzt.
2013	Im Rahmen einer weiteren Überarbeitung wird die dritte Version der IEC 60848:2013 „GRAFCET-specification language for sequential function charts“ veröffentlicht.
2013	Im Rahmen einer weiteren Überarbeitung wird die dritte Version der IEC 61131-3:2013 „Programmable Controllers-Part 3: Programming Languages“ veröffentlicht.

Anhang B Aktionsbestimmungszeichen gemäß DIN EN 61131-3

Grafische Repräsentation	Dynamisches Verhalten	Bemerkung
		<p>nicht-gespeicherte Aktion (<i>Non-stored</i>)</p>
		<p>gespeicherte Aktion (<i>Set stored</i>)</p>
		<p>vorrangig rücksetzende Aktion (<i>overriding Reset</i>)</p>
	 <p>Wenn S1 vor Ablauf von t_1 deaktiviert wird, so wird auch Aktion_1 deaktiviert</p>	<p>zeitbegrenzte Aktion (<i>time Limited</i>)</p>
	 <p>Wenn S1 vor Ablauf von t_1 deaktiviert wird, so wird auch Aktion_1 deaktiviert</p>	<p>zeitverzögerte Aktion (<i>time Delayed</i>)</p>

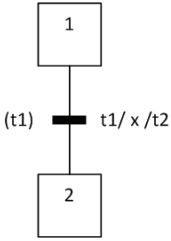
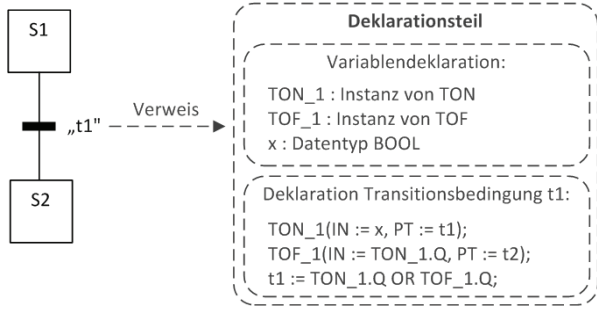
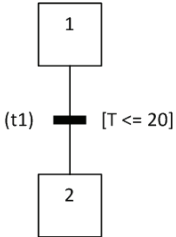
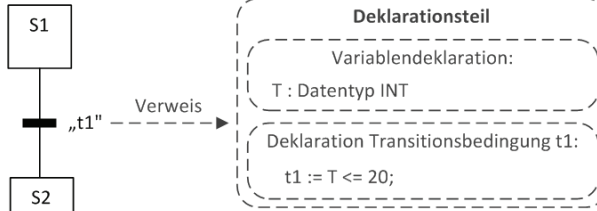
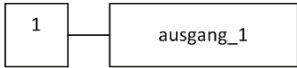
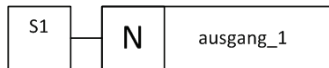
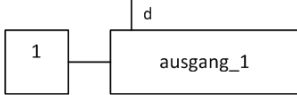
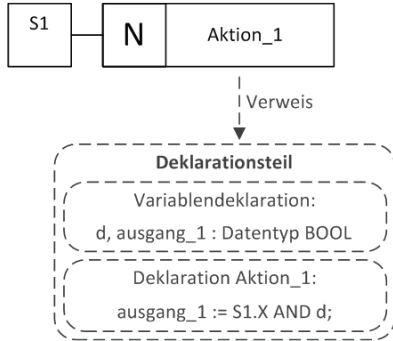

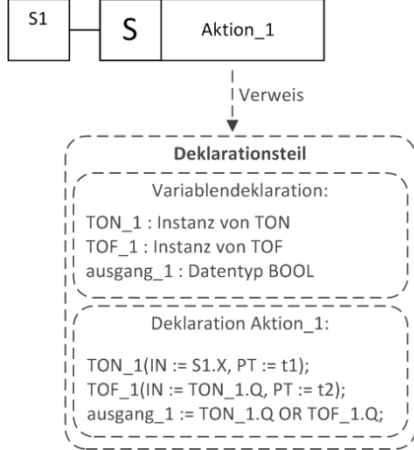
Anhang C Aktionen und Transitionsbedingungen in GRAFCET

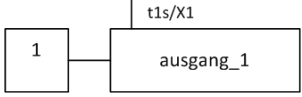
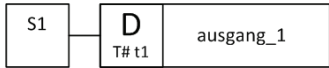
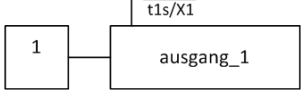
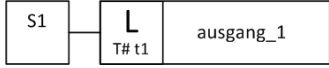
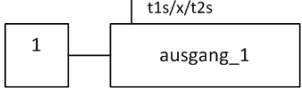
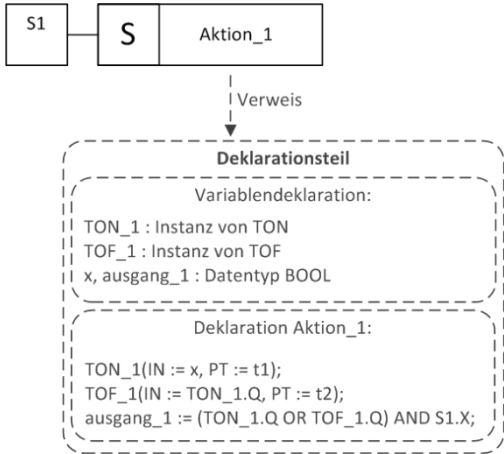
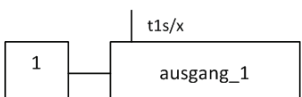
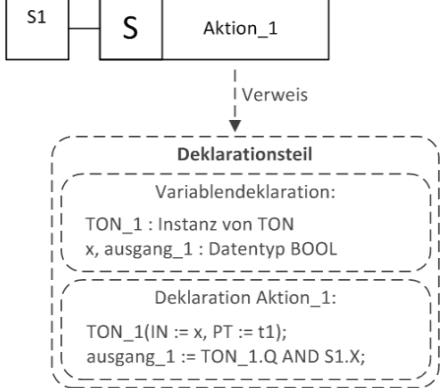
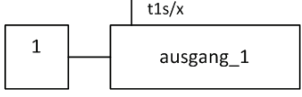
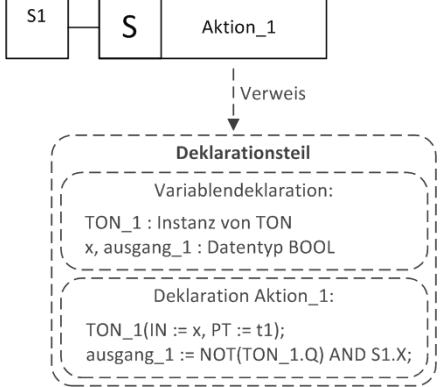
Lfd. Nr.	Grafische Repräsentation	Dynamisches Verhalten	Bemerkung
1		<p>„Das Symbol „1“ bedeutet, dass die Transitionsbedingung immer erfüllt (TRUE) ist.“ [IEC60848].</p>	Immer erfüllte Transitionsbedingung
2			Transitionsbedingung mit Booleschem Ausdruck
3			Transitionsbedingung mit steigender Flanke (↑) / fallender Flanke (↓) einer logischen Variablen
4			Transitionsbedingung mit einer Aussage
5			Zeitabhängige Transitionsbedingung (allgemeiner Fall)
6			Zeitabhängige Transitionsbedingung (spezieller Fall für $t_2 = 0$)

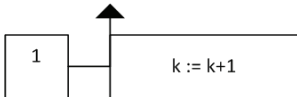
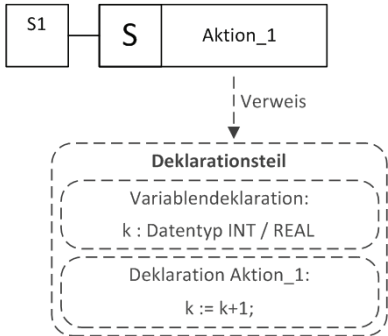
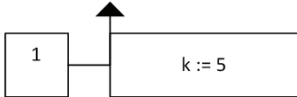
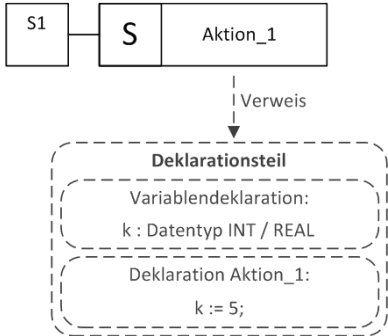
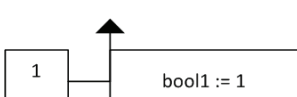
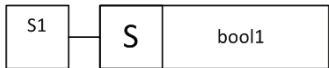
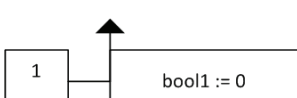
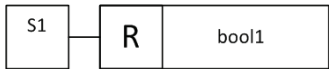
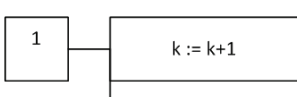
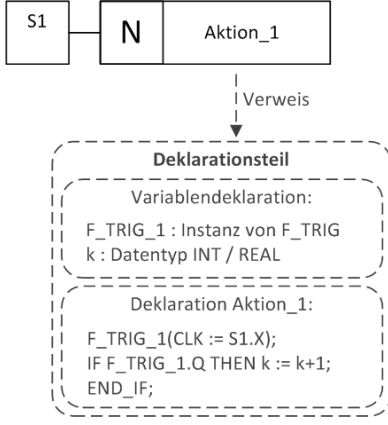
Lfd. Nr.	Grafische Repräsentation	Dynamisches Verhalten	Bemerkung
7			Kontinuierlich wirkende Aktion
8			Kontinuierlich wirkende Aktion mit Zuweisungsbedingung
9			Kontinuierlich wirkende Aktion mit zeitabhängiger Zuweisungsbedingung
10			Speichernd wirkende Aktion bei Aktivierung
11			Speichernd wirkende Aktion bei Deaktivierung
12			Speichernd wirkende Aktion bei Ereignis

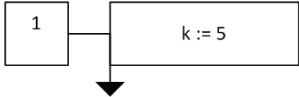
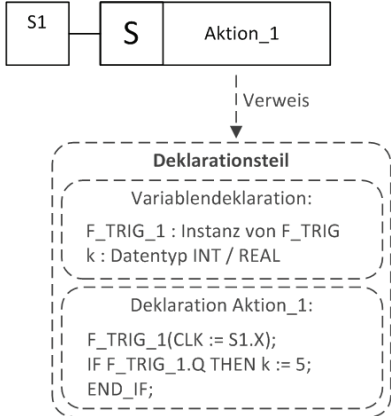
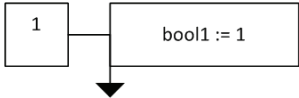
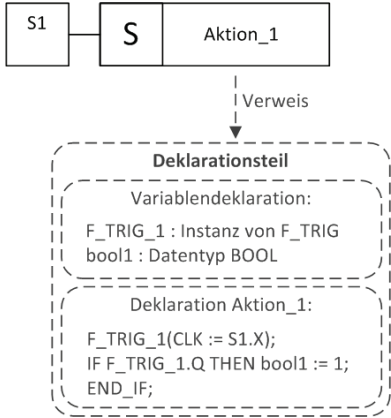
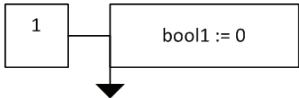
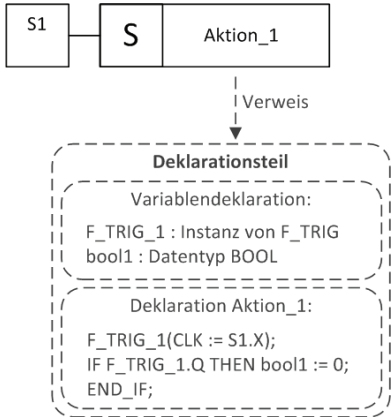
Anhang D Transformationsregeln

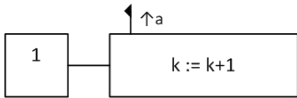
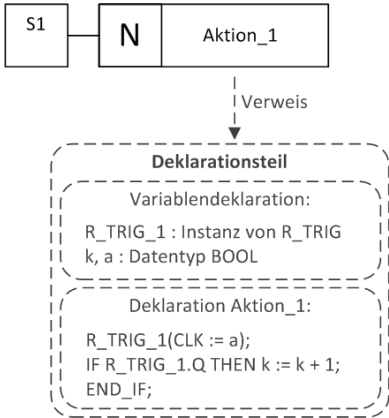
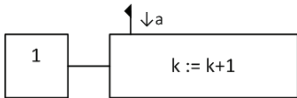
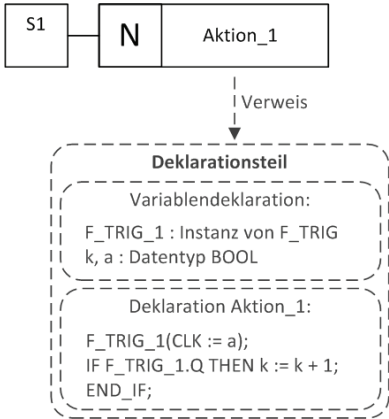
Lfd. Nr.	Element / Konstrukt in GRAFCET	Element / Konstrukt gemäß IEC 61131-3
1 (Schritt)		
2 (Initialschritt)		
3 (Transition)		
4 (Wirkverbindung)		
5 (parallele Verzweigung)		
6 (Transitionsbedingung, Bedingung)		<div style="border: 1px dashed black; padding: 5px; margin-top: 10px;"> <p style="text-align: center;">Deklarationsteil</p> <p>Variablendeklaration: a, b : Datentyp BOOL</p> <p>Deklaration Transitionsbedingung t1: t1 := a AND b;</p> </div>
7 (Transitionsbedingung, Ereignis)		<div style="border: 1px dashed black; padding: 5px; margin-top: 10px;"> <p style="text-align: center;">Deklarationsteil</p> <p>Variablendeklaration: R_TRIG_1 : Instanz von R_TRIG a : Datentyp BOOL</p> <p>Deklaration Transitionsbedingung t1: R_TRIG_1(CLK := a); t1 := R_TRIG_1.Q;</p> </div>
8 (Transitionsbedingung, Ereignis)		<div style="border: 1px dashed black; padding: 5px; margin-top: 10px;"> <p style="text-align: center;">Deklarationsteil</p> <p>Variablendeklaration: F_TRIG_1 : Instanz von F_TRIG a : Datentyp BOOL</p> <p>Deklaration Transitionsbedingung t1: F_TRIG_1(CLK := a); t1 := F_TRIG_1.Q;</p> </div>

Lfd. Nr.	Element / Konstrukt in GRAFCET	Element / Konstrukt gemäß IEC 61131-3
<p>9 (zeitabhängige Transitionsbedingung)</p>		
<p>10 (Transitionsbedingung, logische Aussage)</p>		
<p>11 (kontinuierlich wirkende Aktion)</p>		
<p>12 (kontinuierlich wirkende Aktion mit Zuweisungsbedingung)</p>		
<p>13 (kontinuierlich wirkende Aktion mit zeitabhängiger Bedingung)</p>		

Lfd. Nr.	Element / Konstrukt in GRAFCET	Element / Konstrukt gemäß IEC 61131-3
<p>14 (kontinuierlich wirkende Aktion mit Zeitverzögerung)</p>		
<p>15 (kontinuierlich wirkende Aktion mit Zeitbegrenzung)</p>		
<p>16 (kontinuierlich wirkende Aktion mit zeitabhängiger Bedingung)</p>		
<p>17 (kontinuierlich wirkende Aktion mit zeitabhängiger Bedingung)</p>		
<p>18 (kontinuierlich wirkende Aktion mit zeitabhängiger Bedingung)</p>		

Lfd. Nr.	Element / Konstrukt in GRAFCET	Element / Konstrukt gemäß IEC 61131-3
<p>19 (gespeichert wirkende Aktion bei Schrittaktivierung)</p>		
<p>20 (gespeichert wirkende Aktion bei Schrittaktivierung)</p>		
<p>21 (gespeichert wirkende Aktion bei Schrittaktivierung)</p>		
<p>22 (gespeichert wirkende Aktion bei Schrittaktivierung)</p>		
<p>23 (gespeichert wirkende Aktion bei Schrittdeaktivierung)</p>		

Lfd. Nr.	Element / Konstrukt in GRAFCET	Element / Konstrukt gemäß IEC 61131-3
<p>24 (gespeichert wirkende Aktion bei Schrittdeaktivierung)</p>		
<p>25 (gespeichert wirkende Aktion bei Schrittdeaktivierung)</p>		
<p>26 (gespeichert wirkende Aktion bei Schrittdeaktivierung)</p>		

Lfd. Nr.	Element / Konstrukt in GRAFCET	Element / Konstrukt gemäß IEC 61131-3
<p style="text-align: center;">27 (Aktion bei Ereignis)</p>		
<p style="text-align: center;">28 (Aktion bei Ereignis)</p>		

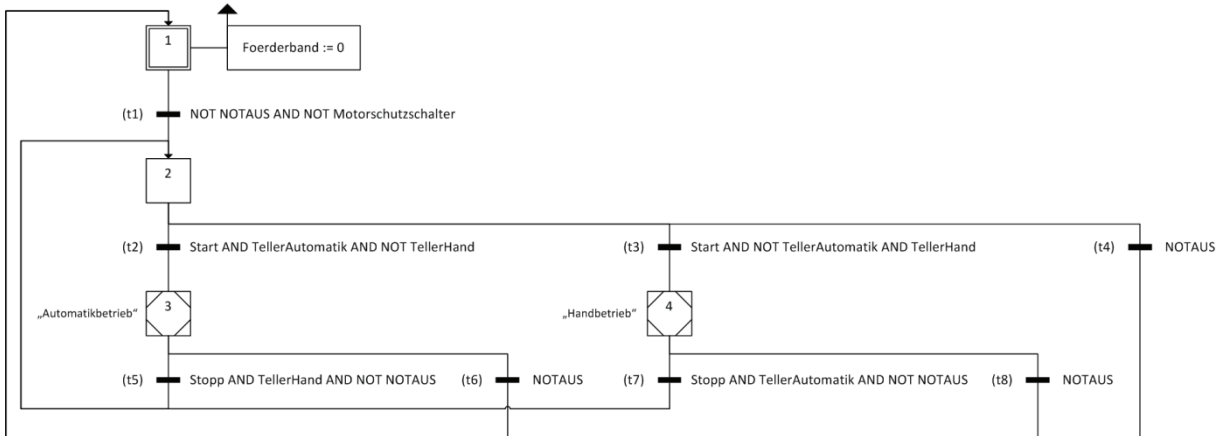
Anhang E Signalübersicht des Prüfautomaten

Anlagen-Teil	Signalname	Signaltyp	Datentyp	Kommentar
Rundschalttisch	NOTAUS	Input	bool	Not-Aus (0, wenn betätigt)
	Start	Input	bool	Start-Taster (1, wenn betätigt)
	Stopp	Input	bool	Stopp-Taster (1, wenn betätigt)
	TellerInPosition	Input	bool	Rundschalttisch ist in Position
	TellerDreht	Input	bool	Rundschalttisch dreht (0, wenn dreht)
	TellerAutomatik	Input	bool	Rundschalttisch in Automatikbetrieb
	TellerHand	Input	bool	Rundschalttisch in Handbetrieb
	StartTeller	Output	bool	Rundschalttisch starten (Drehung)
	Station1_start	Merker	bool	(internes) Startsignal für Station 1
	Station2_start	Merker	bool	(internes) Startsignal für Station 2
	Station3_start	Merker	bool	(internes) Startsignal für Station 3
	Station5_start	Merker	bool	(internes) Startsignal für Station 5
	Station6_start	Merker	bool	(internes) Startsignal für Station 6
	Station7_start	Merker	bool	(internes) Startsignal für Station 7
Station 1	LinearHinten1	Input	bool	Lineareinheit ist in hinterer Position
	LinearVorne1	Input	bool	Lineareinheit ist in vorderer Position
	HandlingOben1	Input	bool	Greifer ist in oberer Position
	HandlingUnten1	Input	bool	Greifer ist in unterer Position
	ZangeAuf1	Input	bool	Greiferzange ist geöffnet
	ZangeZu1	Input	bool	Greiferzange ist geschlossen
	TeilInVereinzelung1	Input	bool	Prüfteil in Vereinzelung
	TeilEingelaufen1	Input	bool	Prüfteil im Übergabe-Bereich
	VorvereinzelungOben1	Input	bool	Vorvereinzelung in oberer Position
	VorvereinzelungUnten1	Input	bool	Vorvereinzelung in unterer Position
	VereinzelungUnten1	Input	bool	Vereinzelung in unterer Position
	VereinzelungOben1	Input	bool	Vereinzelung in oberer Position
	Foerderband	Output	bool	Förderband einschalten
	VorVereinzelung1	Output	bool	Vorvereinzelung runter (wenn betätigt)
	Vereinzelung1	Output	bool	Vereinzelung hoch (wenn betätigt)
	Lineareinheit1	Output	bool	Lineareinheit zu Station 1 fahren
	Handling1	Output	bool	Handling-Einheit runter (wenn betätigt)
	Zange1	Output	bool	Handling-Zange schließen
	Station1_fertig	Merker	bool	= 1, wenn Bearbeitungsvorgang fertig
Station 2	EindrueckenOben2	Input	bool	Kolben in oberer Position
	EindrueckenUnten2	Input	bool	Kolben in unterer Position
	Eindruecken2	Output	bool	Teil Eindrücken runter
	Störung2	Merker	bool	Station 2: Störung
	Station2_fertig	Merker	bool	= 1, wenn Bearbeitungsvorgang fertig
Station 3	SpannenOben3	Input	bool	Spannen in oberer Position
	SpannenUnten3	Input	bool	Spannen in unterer Position
	StoesselOben3	Input	bool	Schaltstößel in oberer Position
	StoesselUnten3	Input	bool	Schaltstößel in unterer Position
	AusloeserHinten3	Input	bool	Auslößer in hinterer Position
	AusloeserVorne3	Input	bool	Auslößer in vorderer Position
	Spannen3	Output	bool	Spannzylinder runter
	Stoessel3	Output	bool	Schaltstößel runter
	Ausloeser3	Output	bool	Auslöser zur Station 3 fahren
	Station3_fertig	Merker	bool	= 1, wenn Bearbeitungsvorgang fertig
Station 5	AusloeserHinten5	Input	bool	Auslößer in hinterer Position
	AusloeserVorne5	Input	bool	Auslößer in vorderer Position
	SpannenOben5	Input	bool	Spannen in oberer Position
	SpannenUnten5	Input	bool	Spannen in unterer Position

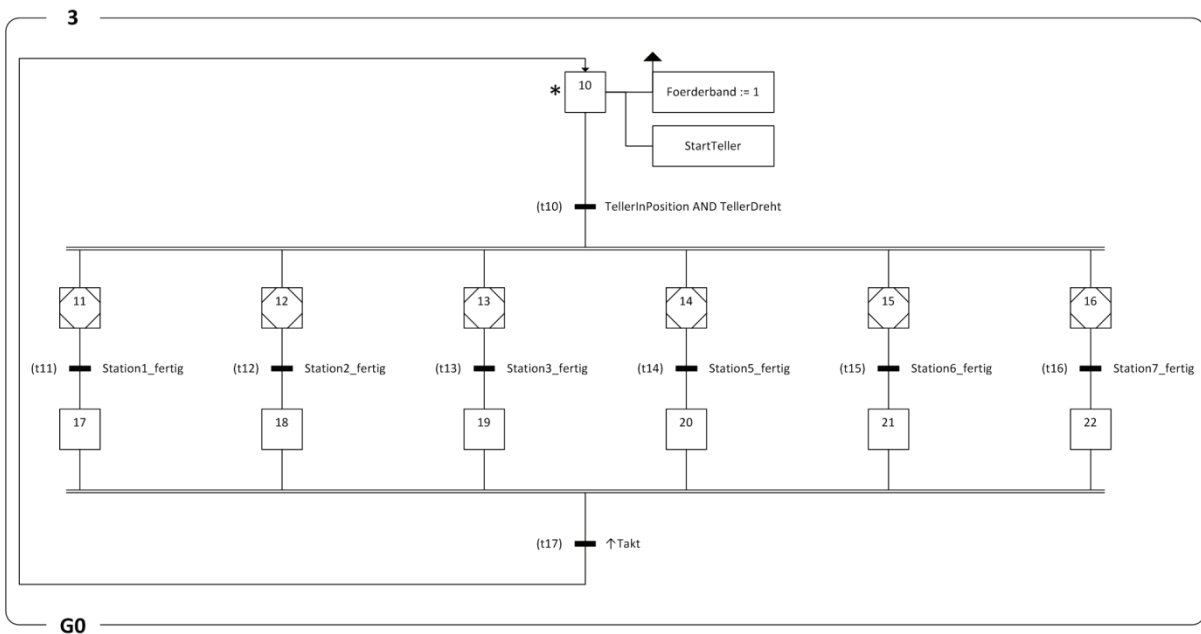
Anlagen-Teil	Signalname	Signaltyp	Datentyp	Kommentar
	StoesselOben5	Input	bool	Schaltstößel in oberer Position
	StoesselUnten5	Input	bool	Schaltstößel in unterer Position
	Kontrolle1	Input	bool	Kontrolle der Kontaktierung 1 erfolgt
	Kontrolle2	Input	bool	Kontrolle der Kontaktierung 2 erfolgt
Station 5	KontaktierungHinten5	Input	bool	Kontaktierung in hinterer Position
	KontaktierungVorne5	Input	bool	Kontaktierung in vorderer Position
	Spannen5	Output	bool	Spannzylinder runter
	Stoessel5	Output	bool	Schaltstößel runter
	Ausloeser5	Output	bool	Auslöser zu Station 5 fahren
	Kontaktierung5	Output	bool	Kontaktierung zu Station 5 fahren
	GUTTEIL	Merker	bool	= 1 für Gutteil; = 0 für Schlechtteil
	Station5_fertig	Merker	bool	= 1, wenn Bearbeitungsvorgang fertig
Station 6	StempelHinten6	Input	bool	Stempel in hinterer Position
	StempelVorne6	Input	bool	Stempel in vorderer Position
	Stempeln6	Output	bool	Stempelzylinder zu Station 6 fahren
	Station6_fertig	Merker	bool	= 1, wenn Bearbeitungsvorgang fertig;
Station 7	LinearHinten7	Input	bool	Lineareinheit in hinterer Position
	LinearVorne7	Input	bool	Lineareinheit in vorderer Position
	HandlingOben7	Input	bool	Greifer in oberer Position
	HandlingUnten7	Input	bool	Greifer in unterer Position
	ZangeAuf7	Input	bool	Greiferzange geöffnet
	ZangeZu7	Input	bool	Greiferzange geschlossen
	LinearInPos7	Input	bool	Linearantrieb in "Schlecht"-Position
	TeilAufTeller7	Input	bool	Werkstück am Übergabepunkt vom Runddrehtisch zu Station 7
	LineareinheitVor7	Output	bool	Lineareinheit vorfahren
	LineareinheitZur7	Output	bool	Lineareinheit zurückfahren
	Handling7	Output	bool	Handling-Einheit runter
	Zange7	Output	bool	Handling-Zange schließen
	Station7_fertig	Merker	bool	= 1, wenn Bearbeitungsvorgang fertig

Anhang F GRAFCET-Spezifikation des Prüfautomaten

Globaler Grafcet der Betriebsarten



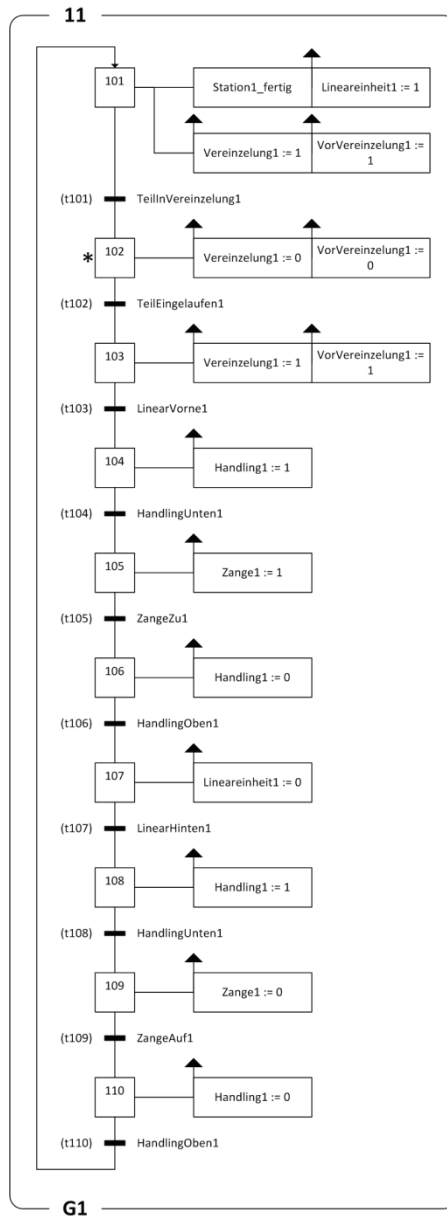
Teil-Grafcet G0 (Runddrehtisch im Automatikbetrieb)



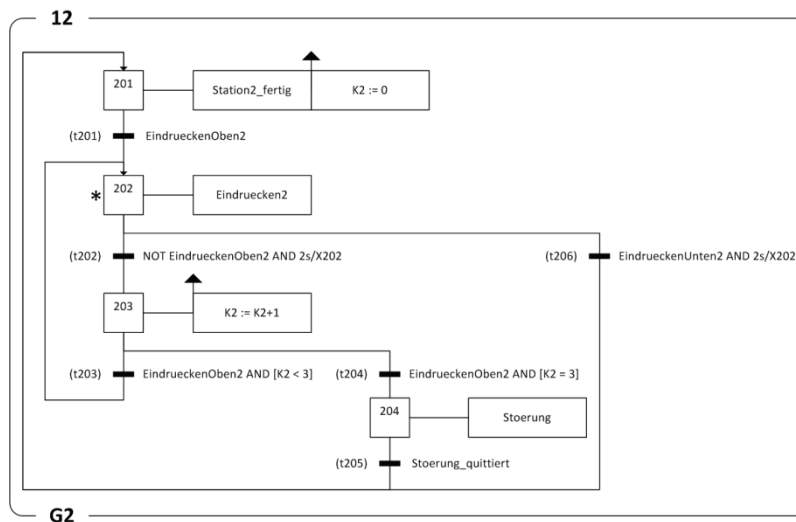
Das in der Transition t_{17} enthaltene Ereignis $\uparrow Takt$ ist die abgekürzte Schreibweise für:

$$\uparrow Takt = \uparrow (Station1_fertig \wedge Station2_fertig \wedge Station3_fertig \wedge Station5_fertig \wedge Station6_fertig \wedge Station7_fertig)$$

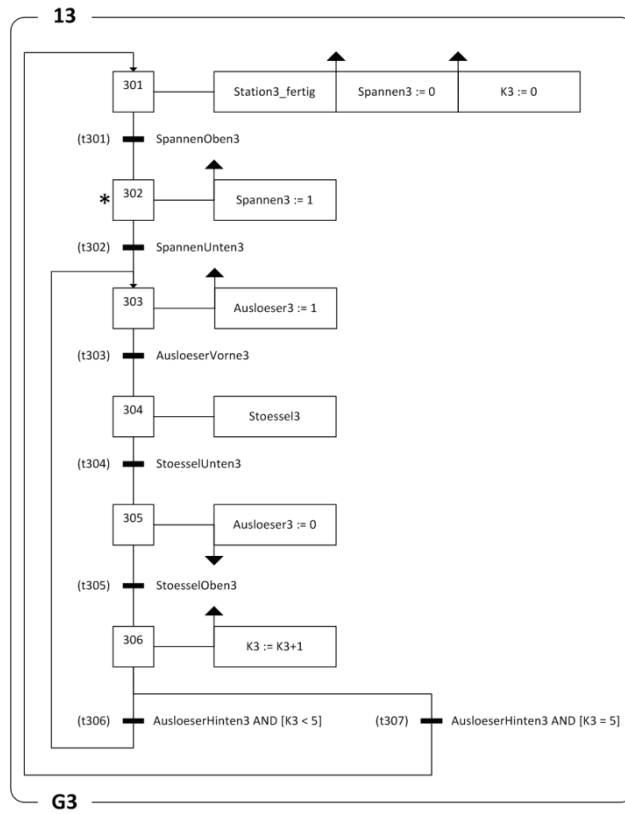
Teil-Grafset G1 (Station 1 im Automatikbetrieb)



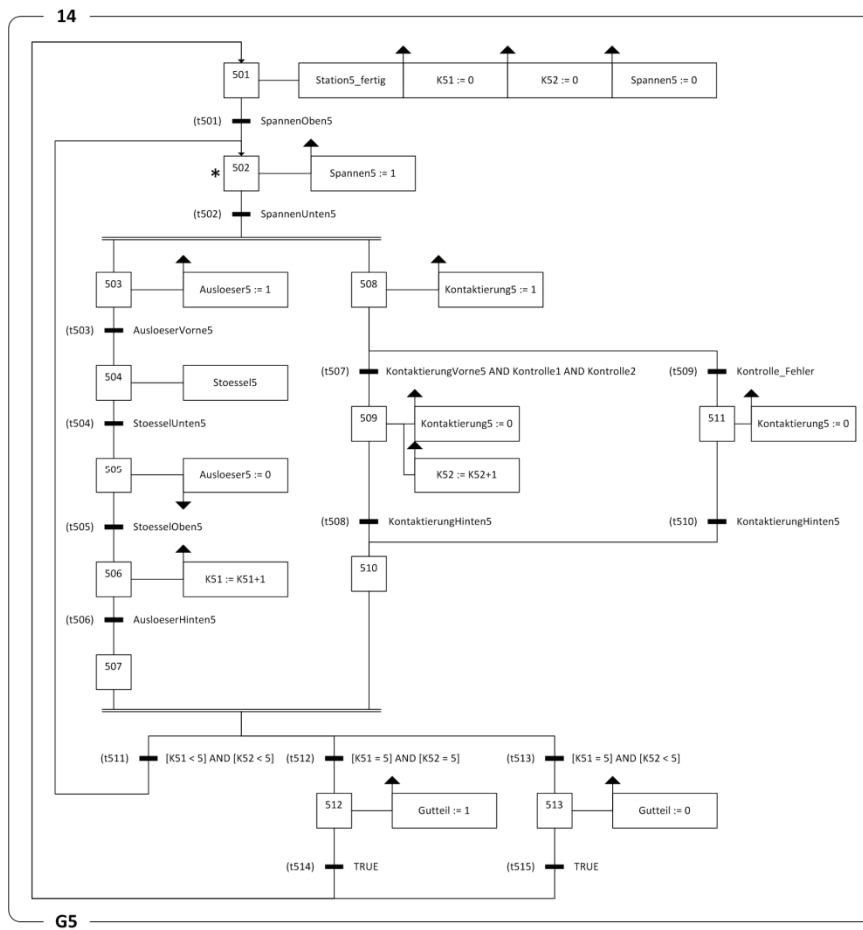
Teil-Grafset G2 (Station 2 im Automatikbetrieb)



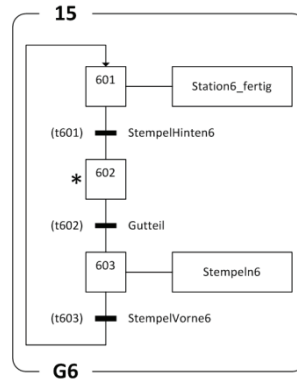
Teil-Grafset G3 (Station 3 im Automatikbetrieb)



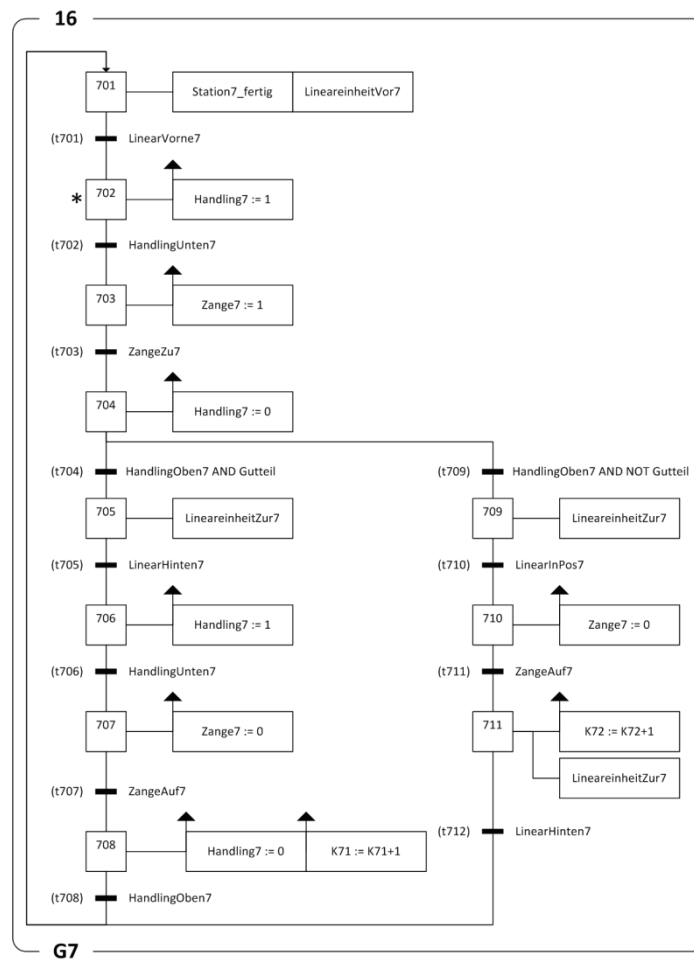
Teil-Grafset G5 (Station 5 im Automatikbetrieb)



Teil-Grafset G6 (Station 6 im Automatikbetrieb)



Teil-Grafset G7 (Station 7 im Automatikbetrieb)



Anhang G Struktur des automatisch generierten SFC des Prüfautomaten

INITIAL_STEP INIT: END_STEP
STEP Step1: Foerderband (R); END_STEP
STEP Step2: StartTeller (N);Foerderband (S); END_STEP
STEP Step3: END_STEP
STEP Step4: END_STEP
STEP Step5: END_STEP
STEP Step6: END_STEP
STEP Step7: END_STEP
STEP Step8: END_STEP
STEP Step9: END_STEP
STEP Step10: END_STEP
STEP Step11: END_STEP
STEP Step12: END_STEP
STEP Step13: END_STEP
STEP Step14: END_STEP
STEP Step101: Station1_fertig (N);Lineareinheit1 (S);Vereinzelung1 (S);VorVereinzelung1 (S); END_STEP
STEP Step102: Vereinzelung1 (R);VorVereinzelung1 (R); END_STEP
STEP Step103: Vereinzelung1 (S);VorVereinzelung1 (S); END_STEP
STEP Step104: Handling1 (S); END_STEP
STEP Step105: Zange1 (S); END_STEP
STEP Step106: Handling1 (R); END_STEP
STEP Step107: Lineareinheit1 (R); END_STEP
STEP Step108: Handling1 (S); END_STEP
STEP Step109: Zange1 (R); END_STEP
STEP Step110: Handling1 (R); END_STEP
STEP Step201: Station2_fertig (N);Action_1 (S); END_STEP
STEP Step202: Eindruecken2 (N); END_STEP
STEP Step203: Action_2 (S); END_STEP
STEP Step204: Stoerung (N); END_STEP
STEP Step301: Station3_fertig (N);Spannen3 (R);Action_3 (S); END_STEP
STEP Step302: Spannen3 (S); END_STEP
STEP Step303: Ausloeser3 (S); END_STEP
STEP Step304: Stoessel3 (N); END_STEP
STEP Step305: Action_4 (S); END_STEP
STEP Step306: Action_5 (S); END_STEP
STEP Step501: Station5_fertig (N);Action_6 (S);Action_7 (S);Spannen5 (S); END_STEP
STEP Step502: Spannen5 (S); END_STEP
STEP Step503: Ausloeser5 (S); END_STEP
STEP Step504: Stoessel5 (N); END_STEP
STEP Step505: Action_8 (S); END_STEP
STEP Step506: Action_9 (S); END_STEP
STEP Step507: END_STEP
STEP Step508: Kontaktierung5 (S); END_STEP
STEP Step509: Kontaktierung5 (R);Action_10 (S); END_STEP
STEP Step510: END_STEP
STEP Step511: Kontaktierung5 (R); END_STEP
STEP Step512: Gutteil (S); END_STEP
STEP Step513: Gutteil (R); END_STEP
STEP Step601: Station6_fertig (N); END_STEP
STEP Step602: END_STEP
STEP Step603: Stempeln6 (N); END_STEP
STEP Step701: Station7_fertig (N);LineareinheitVor7 (N); END_STEP
STEP Step702: Handling7 (S); END_STEP
STEP Step703: Zange7 (S); END_STEP
STEP Step704: Handling7 (R); END_STEP
STEP Step705: LineareinheitZur7 (N); END_STEP
STEP Step706: Handling7 (S); END_STEP
STEP Step708: Zange7 (R); END_STEP

STEP Step709: Handling7 (R);Action_11 (S); END_STEP
STEP Step710: LineareinheitZur7 (N); END_STEP
STEP Step711: Zange7 (R); END_STEP
STEP Step712: Action_12 (S);LineareinheitZur7 (N); END_STEP
TRANSITION FROM INIT TO Step1 := t_INIT ; END_TRANSITION
TRANSITION FROM Step1 TO Step2 := t1 ; END_TRANSITION
TRANSITION FROM Step2 TO (Step3, Step4, Step5, Step6, Step7, Step8, Step102, Step202, Step302, Step502, Step602, Step702) := t2 ; END_TRANSITION
TRANSITION FROM Step3 TO Step9 := t3 ; END_TRANSITION
TRANSITION FROM Step4 TO Step10 := t4 ; END_TRANSITION
TRANSITION FROM Step5 TO Step11 := t5 ; END_TRANSITION
TRANSITION FROM Step6 TO Step12 := t6 ; END_TRANSITION
TRANSITION FROM Step7 TO Step13 := t7 ; END_TRANSITION
TRANSITION FROM Step8 TO Step14 := t8 ; END_TRANSITION
TRANSITION FROM (Step9, Step10, Step11, Step12, Step13, Step14) TO Step2 := t9 ; END_TRANSITION
TRANSITION FROM (Step9, Step10, Step11, Step12, Step13, Step14) TO Step1 := t10 ; END_TRANSITION
TRANSITION FROM Step101 TO Step102 := t101 ; END_TRANSITION
TRANSITION FROM Step102 TO Step103 := t102 ; END_TRANSITION
TRANSITION FROM Step103 TO Step104 := t103 ; END_TRANSITION
TRANSITION FROM Step104 TO Step105 := t104 ; END_TRANSITION
TRANSITION FROM Step105 TO Step106 := t105 ; END_TRANSITION
TRANSITION FROM Step106 TO Step107 := t106 ; END_TRANSITION
TRANSITION FROM Step107 TO Step108 := t107 ; END_TRANSITION
TRANSITION FROM Step108 TO Step109 := t108 ; END_TRANSITION
TRANSITION FROM Step109 TO Step110 := t109 ; END_TRANSITION
TRANSITION FROM Step110 TO Step101 := t110 ; END_TRANSITION
TRANSITION FROM Step201 TO Step202 := t201 ; END_TRANSITION
TRANSITION FROM Step202 TO Step203 := t202 ; END_TRANSITION
TRANSITION FROM Step203 TO Step202 := t203 ; END_TRANSITION
TRANSITION FROM Step202 TO Step201 := t206 ; END_TRANSITION
TRANSITION FROM Step203 TO Step204 := t204 ; END_TRANSITION
TRANSITION FROM Step204 TO Step201 := t205 ; END_TRANSITION
TRANSITION FROM Step301 TO Step302 := t301 ; END_TRANSITION
TRANSITION FROM Step302 TO Step303 := t302 ; END_TRANSITION
TRANSITION FROM Step303 TO Step304 := t303 ; END_TRANSITION
TRANSITION FROM Step304 TO Step305 := t304 ; END_TRANSITION
TRANSITION FROM Step305 TO Step306 := t305 ; END_TRANSITION
TRANSITION FROM Step306 TO Step303 := t306 ; END_TRANSITION
TRANSITION FROM Step306 TO Step301 := t307 ; END_TRANSITION
TRANSITION FROM Step501 TO Step502 := t501 ; END_TRANSITION
TRANSITION FROM Step502 TO (Step503, Step508) := t502 ; END_TRANSITION
TRANSITION FROM Step503 TO Step504 := t503 ; END_TRANSITION
TRANSITION FROM Step504 TO Step505 := t504 ; END_TRANSITION
TRANSITION FROM Step505 TO Step506 := t505 ; END_TRANSITION
TRANSITION FROM Step506 TO Step507 := t506 ; END_TRANSITION
TRANSITION FROM Step508 TO Step509 := t507 ; END_TRANSITION
TRANSITION FROM Step509 TO Step510 := t508 ; END_TRANSITION
TRANSITION FROM Step508 TO Step511 := t509 ; END_TRANSITION
TRANSITION FROM Step511 TO Step510 := t510 ; END_TRANSITION
TRANSITION FROM (Step507, Step510) TO Step502 := t511 ; END_TRANSITION
TRANSITION FROM (Step507, Step510) TO Step512 := t512 ; END_TRANSITION
TRANSITION FROM (Step507, Step510) TO Step513 := t513 ; END_TRANSITION
TRANSITION FROM Step512 TO Step501 := t514 ; END_TRANSITION
TRANSITION FROM Step513 TO Step501 := t515 ; END_TRANSITION
TRANSITION FROM Step602 TO Step603 := t602 ; END_TRANSITION
TRANSITION FROM Step603 TO Step601 := t603 ; END_TRANSITION
TRANSITION FROM Step601 TO Step602 := t601 ; END_TRANSITION
TRANSITION FROM Step701 TO Step702 := t701 ; END_TRANSITION
TRANSITION FROM Step702 TO Step703 := t702 ; END_TRANSITION
TRANSITION FROM Step703 TO Step704 := t703 ; END_TRANSITION
TRANSITION FROM Step704 TO Step705 := t704 ; END_TRANSITION
TRANSITION FROM Step704 TO Step709 := t709 ; END_TRANSITION

Literaturverzeichnis

Das Literaturverzeichnis enthält nicht vom Verfasser stammende Quellen.
Sie werden mit [<KURZBELEG>] referenziert.

- [ALUR ET AL. 1996] **Alur, R., Henzinger, T. A., Ho, P. H.:**
Automatic symbolic verification of embedded systems.
In: IEEE Transactions on Software Engineering 22, Heft 3/1996, S. 181-201.
- [ALVAREZ ET AL. 2012] **Alvarez, M.L, Burgos, A., Sarachaga, I., Estévez, E., Marcos, M.:**
GEMMA based approach for generating PLCopen Automation projects.
In: Tagungsband "IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control", Würzburg, 03.-05.04.2012, S. 230-235.
- [AUER 1991] **Auer, A.:**
Speicherprogrammierbare Steuerungen, SPS, Aufbau und Programmierung,
Band 1, 4. überarbeitete und erweiterte Auflage, Hüthig-Verlag Heidelberg,
1991, S. 20.
- [ARNOLD & HENRIQUES 2005] **Arnold, G.V., Henriques, P.R.:**
A Graphical Interface Based on GRAFCET for programming Industrial Robots off-line.
In: Tagungsband "International Conference on Informatics in Control, Automation and Robotics", Barcelona (Spanien), 14.-17.09.2005, S. 113-118.
- [AZEVEDO & ESTIMA DE OLIVEIRA 1999] **Azevedo, J. L., Estima de Oliveira, J. P.:**
The Grafcet's macro-action concept: an implementation view.
In: Tagungsband "7th IEEE Conference on Emerging Technologies and Factory Automation", Barcelona (Spanien), 18.-21.10.1999, S. 1275-1279.
- [BAKER ET AL. 1987] **Baker, A. D., Johnson, T. L., Kerpelmann, D. L., Sutherland, H. A.:**
GRAFCET and SFC as Factory Automation Standards-Advantages and Limitations.
In: Tagungsband "American Control Conference", Minneapolis (USA), 10.-12.06.1987, S. 1725-1730.
- [BANI YOUNIS 2006] **Bani Younis, M.:**
Re-Engineering Approach for PLC Programs based on Formal Methods.
Dissertation im Fachbereich Elektro- und Informationstechnik der Technischen Universität Kaiserslautern, Kaiserslautern, 2006.
- [BARESI ET AL. 2000] **Baresi, L., Mauri, M., Monti, A., Pezze, M.: PLCTools:**
Design, Formal Validation, and Code Generation for Programmable Controllers.
In: Tagungsband "IEEE International Conference on Systems, Man and Cybernetics", Nashville (USA), 08.-11.10.2000, S. 2437-2442.
- [BARRAGAN SANTIAGO & FAURE 2005] **Barragan Santiago, I., Faure, J.-M.:**
From fault tree analysis to model checking of logic controllers.
In: Tagungsband "16th IFAC World Congress", Prag (Tschechische Republik), 03.-08.07.2005, CD-ROM Beitrag Nr. 4596.
- [BARTH & FAY 2010] **Barth, M., Fay, A.:**
Efficient use of data exchange formats in engineering projects by means of language integrated queries - Engineers LINQ to XML.
In: Tagungsband "36th Annual Conference on IEEE Industrial Electronics Society", Glendale (USA), 07.-10.11.2010, S. 1335-1340.

- [BAUER 2002] **Bauer, N.:**
Statecharts versus Sequential Function Charts.
In: Automatisierungstechnik - at 50, Heft 11/2002, S. 533-540.
- [BAUER 2003] **Bauer, N.:**
Formale Analyse von Sequential Function Charts.
Dissertation an der Universität Dortmund, Fachbereich Chemietechnik,
Dortmund, 2003.
- [BAUER ET AL. 2002] **Bauer, N., Huuck, R., Lukoschus, B.:**
A stopwatch semantics for hybrid controllers.
In: Tagungsband "IFAC Triennial World Congress", Barcelona (Spanien), 21.-
26.07.2002.
- [BAUER ET AL. 2003] **Bauer, N., Engell, S., Lohmann, S.:**
Vergleich toolspezifischer Implementierungen von Sequential Function Charts (SFCs). In: Tagungsband "SPS / IPC / Drives", Nürnberg, 25.-27.07.2003, S. 215-224.
- [BAUER ET AL. 2004 A] **Bauer, N., Huuck, R., Lukoschus, B., Engell, S.:**
A Unifying Semantics for Sequential Function Charts.
In: Integration of Software Specification Techniques for Applications in
Engineering, Band 3147, S. 400-418.
- [BAUER ET AL. 2004 B] **Bauer, N., Huuck, R., Lohmann, S., Lukoschus, B.:**
Sequential Function Charts: Die Notwendigkeit formaler Analyse.
In: Automatisierungstechnische Praxis - atp 46, Heft 8/2004, S. 61-67.
- [BAYÓ-PUXAN ET AL. 2008] **Bayo-Puxan, O., Rafecas-Sabaté, J., Gomis-Bellmunt, O., Bergas-Jané, J.:**
A GRAFCET-compiler methodology for C-programmed microcontrollers.
In: Assembly Automation 28, Heft 1/2008, S. 55-60.
- [BAYRAK ET AL. 2008] **Bayrak, G., Abrishamchian, F., Vogel-Heuser, B.:**
*Effiziente Steuerungsprogrammierung durch automatische
Modelltransformation von Matlab/Simulink/Stateflow nach IEC 61131-3.*
In: Automatisierungstechnische Praxis - atp 50, Heft 12/2008, S. 49-55.
- [BLANCHARD 1979] **Blanchard, M.:**
Comprendre maîtriser et appliquer le GRAFCET. éditions CEPADUES,
Collection NABLA, 1979.
- [CARRÉ-MÉNÉTRIER ET AL. 1999] **Carre-Menetrier, V., Ndjab, C., Gellot, F., Zaytoon, J.:**
A CASE tool for the synthesis of optimal control implementation of Grafset.
In: Tagungsband "International Conference on Systems, Man and Cybernetics",
Tokyo (Japan), 12.-15.10.1999, S. 796-801.
- [CARRÉ-MÉNÉTRIER & ZAYTOON 2002] **Carré-Ménétrier, V., Zaytoon, J.:**
Grafset: Behavioural Issues and Control Synthesis.
In: European Journal of Control 8, Heft 4 / 2002, S. 375-401.
- [CHARBONNIER ET AL. 1995] **Charbonnier, F., Alla, H., David, R.:**
The supervised control of discrete event dynamic systems: a new approach.
In: Tagungsband "IEEE Conference on Decision and Control", New Orleans
(USA), 13.-15.12.1995, S. 913-920.
- [CHÉRIAUX ET AL. 2010] **Chériaux, F., Picci, L., Provost, J., Faure, J. M.:**
*Conformance test of logic controllers of critical systems from industrial
specifications.* In: Tagungsband "European Conference on Safety and
Reliability", Rhodos (Griechenland), 05.-09.09.2010, S. 1569-1576.

- [CHRISTIANSEN ET AL. 2011] **Christiansen, L., Jäger, T., Strube, M., Fay, A.:**
Integration of a formalized process description into MS Visio® with regard to an integrated engineering process.
In: Tagungsband "Workshop on Industrial Automation Tool Integration for Engineering Project Automation", IEEE International Conference on Emerging Technologies and Factory Automation, Toulouse (Frankreich), 09.09.2011.
- [CLOUTIER & PAQUES 1988] **Cloutier, G., Paques, J.-J.:**
GEMMA, the complementary tool of the GRAFCET.
In: Tagungsband "4th annual Canadian programmable control and automation technology conference and exhibition", Toronto, 12.10.-13.10.1988, S. 12A1-5:1 - 12A1-5:10.
- [DAVID 1995] **David, R.:**
Grafcet: a powerful tool for specification of logic controllers.
In: IEEE Transactions on Control Systems Technology 3, Nr. 3/1995, S. 253-268.
- [DAVID & ALLA 1992] **David, R., Alla, H.:**
Petri Nets and Grafcet, Tools for modeling discrete event systems. Prentice Hall New York, 1992.
- [DAVID & ALLA 2010] **David, R., Alla, H.:**
Discrete, Continuous, and Hybrid Petri Nets. Springer-Verlag Berlin Heidelberg, 2010.
- [DENEUX 1983] **Deneux, H.:**
Contribution à l'étude des réseaux d'automates interconnectés.
Dissertation an der Grenoble University (IPNG), Grenoble (Frankreich), 1983.
- [DEZANI ET AL. 2011] **Dezani, H., Marranghello, N., Pereira, A.S., da Silva, A.C.R., Wegrzyn, M.:**
Automatic Code Generation for Microcontrollers from Place-Transition Petri Net Models.
In: Tagungsband "18th IFAC World Congress", Mailand (Italien), 28.08.-02.09.2011, S. 7873-7878.
- [DIDEBAN & ALLA 2006] **Dideban, A., Alla, H.:**
Solving the problem of forbidden states by feedback control logical synthesis.
In: Tagungsband "32nd Annual Conference in Industrial Electronics", Paris (Frankreich), 06.-10.11.2006, S. 348-353.
- [DRATH ET AL. 2011] **Drath, R., Fay, A., Barth, M.:**
Interoperabilität von Engineering-Werkzeugen – Konzepte und Empfehlungen für den Datenaustausch zwischen Engineering-Werkzeugen.
In: Automatisierungstechnik-at 59, Heft 7/2011, S. 451-459.
- [EL RHALIBI ET AL. 1994] **El Rhalibi, A., Prunet, F., Durante, C.:**
Analysis of function charts for control systems using Petri nets.
In: Tagungsband "IEEE Symposium on Emerging Technologies & Factory Automation", Montpellier (Frankreich), 06.-10.11.1994, S. 365-372.
- [EL RHALIBI ET AL. 1995] **El Rhalibi, A., Prunet, F., Durante, C.:**
From modelling using function charts for control systems to analysis using Petri nets. In: Tagungsband "International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems", Montpellier (Frankreich), 18.-20.01.1995, S. 389-393.
- [EPPLE 2003] **Epple, U.:**
Austausch von Anlagenplanungsdaten auf der Grundlage von Metamodellen.
In: Automatisierungstechnische Praxis - atp 45, Heft 7/2003, S. 61-70.

- [ESTÉVEZ ET AL. 2007] **Estévez, E., Marcos, M., Orive, D.:**
Automatic generation of PLC automation projects from component-based models. In: The International Journal of Advanced Manufacturing Technology 35, Heft 5-6/2007, S. 527-540.
- [FELDMANN ET AL. 1999 A] **Feldmann, K., Colombo, A.W., Schnur, C., Stockel, T.:**
Specification, design, and implementation of logic controllers based on colored Petri net models and the standard IEC 1131 - Part I: Specification and design. In: IEEE Transactions on Control Systems Technology 7, Heft 6/1999, S. 657-665.
- [FELDMANN ET AL. 1999 B] **Feldmann, K., Colombo, A.W., Schnur, C., Stockel, T.:**
Specification, design, and implementation of logic controllers based on colored Petri net models and the standard IEC 1131 - Part II: Design and Implementation. In: IEEE Transactions on Control Systems Technology 7, Heft 6/1999, S. 666-675.
- [FERRARINI 2006] **Ferrarini, L.:**
Modular design and implementation of a logic control system for a batch process. In: Computers and Chemical Engineering 27, Heft 7/2006, S. 983-996.
- [FESTO 2008] *GRAF CET*, 1. Auflage, korrigierter Nachdruck, Bildungsverlag EINS Troisdorf, 2008.
- [FISCHER & ENGELL 2011] **Fischer, S., Engell, S.:**
Werkzeugunterstützung für den Entwurf von Ablaufsteuerungen auf Basis informeller Spezifikationen. In: Automatisierungstechnik - at 59, Heft 1/2011 S. 50-61.
- [FRENSEL & BRUIJN 1998] **Frensel, G., Bruijn, P.M.:**
From modelling control systems using Grafcet to analyzing systems using hybrid automata. In: Tagungsband "American Control Conference", Philadelphia (USA), 21.-26.06.1998, S.704-705
- [FREY & KLEIN 2004] **Frey, G., Klein, S.:**
Spezifikation von Betriebszuständen und Betriebsartenumschaltungen - GEMMA: Ein Ansatz aus Frankreich. In: Automatisierungstechnische Praxis - atp 46, Heft 3/2004, S. 37-40.
- [FREY & LITZ 1997] **Frey, G., Litz, L.:**
Transparenter Steuerungsentwurf mit SFC nach IEC 1131-3. In: Tagungsband "SPS / IPC / Drives", Nürnberg, 25.-27.11.1997, S. 240-249.
- [FREY & LITZ 1998] **Frey, G., Litz, L.:**
Verification and Validation of Control Algorithms by Coupling of Interpreted Petri Nets. In: Tagungsband "IEEE International Conference on Systems, Man, and Cybernetics", San Diego (USA), 11.-14.10.1998, S. 7-12.
- [FREY & LITZ 2000] **Frey, G., Litz, L.:**
Formal methods in PLC programming. In: Tagungsband "IEEE International Conference on Systems, Man, and Cybernetics", Nashville (USA), 08.-11.10.2000, S. 2431-2436.
- [FREY & SCHMIDT 2000] **Frey, G., Schmidt, A.:**
Automatische Erzeugung von SPS-Programmen aus Petrinetzen. In: Tagungsband "Fachtagung Verteilte Automatisierung", Magdeburg, 22.-23.03.2000, S. 177-184.

- [FREY & THRAMBOULIDIS 2011] **Frey, G., Thramboulidis, K.:**
Einbindung der IEC 61131 in modellgetriebene Entwicklungsprozesse.
In: Tagungsband „Automationskongress 2011“, Baden-Baden, 28.-29.06.2011, S. 21-24.
- [FREY 2000] **Frey, G.:**
Automatic Implementation of Petri net based Control Algorithms on PLC.
In: Tagungsband "American Control Conference", Chicago (USA), 28.-30.06.2000, S. 2819-2823.
- [FREY 2002] **Frey, G.:**
Design and formal Analysis of Petri Net based Logic Control Algorithms.
Dissertation im Fachbereich Elektro- und Informationstechnik der Universität Kaiserslautern, Kaiserslautern, 2002.
- [GAERTNER & THIRION 1999] **Gaertner, N., Thirion, B.:**
GRAF CET-an Analysis Pattern for Event Driven Real-time Systems.
In: Tagungsband "Conference on Pattern Languages of Programs", Urbana (USA), 15.-18.08.1999.
- [GREPA 1985] **Groupe Equipement de Production Automatisée réuni à l'ADEPA:**
Le GRAFCET, de nouveaux concepts. éditions CEPADUES, Toulouse (Frankreich), 1985.
- [GUÉGUEN ET AL. 1999] **Guéguen, H., Lefebvre, M. A., Bouteille, N.:**
Une méthode d'analyse par objectifs pour la conception de la commande.
In: Tagungsband "Modélisation des Systèmes Réactifs", Cachan (Frankreich), 1999, S. 177-182.
- [GUÉGUEN & BOUTEILLE 2001] **Guéguen, H., Bouteille, N.:**
Extensions to Grafcet to structure behavioural specifications.
In: Control Engineering Practice 9, Heft 7/2001, S. 743-756.
- [GUILLEMAUD & GUÉGUEN 1999] **Guillemaud, L., Guéguen, H.:**
Extending GRAFCET for the specification of control of hybrid systems.
In: Tagungsband "IEEE Conference on Systems, Man and Cybernetics", Tokyo (Japan), 12.-15.10.1999, Band 1, S. 171-175.
- [GÜTTEL ET AL. 2008] **Güttel, K., Weber, P., Fay, A.:**
Automatic generation of PLC code beyond the nominal sequence.
In: Tagungsband "IEEE International Conference on Emerging Trends and Factory Automation", Hamburg, 15.-18.09.2008, S. 1277-1284.
- [GÜTTEL 2012] **Güttel, K.:**
Konzept zur Generierung von Steuerungscode für Fertigungsanlagen unter Verwendung wissensbasierter Methoden. Dissertation an der Fakultät für Maschinenbau der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg, Hamburg, 2012.
- [HAJARNAVIS & YOUNG 2008] **Hajarnavis, V., Young, K.:**
An investigation into programmable logic controller software design techniques in the automotive industry.
In: Assembly Automation 28, Heft 1/2008, S. 43-54.
- [HANISCH ET AL. 1997] **Hanisch, H.-M., Thieme, J., Lüder, A., Wienhold, O.:**
Modeling of PLC behavior by means of timed net condition/event systems.
In: Tagungsband "6th International Conference on emerging Trends and Factory Automation", Los Angeles (USA), 09.-12.09.1997, S. 391-396.
- [HANISCH ET AL. 1998] **Hanisch, H.-M., Lüder, A., Thieme, J.:**
A modular plant modeling technique and related controller synthesis problems.
In: Tagungsband "IEEE International Conference on Systems, Man and Cybernetics", San Diego (USA), 11.-14.10.1998, S. 686-691.

- [HAPPACHER 2012] **Happacher, M. (Hrsg.):**
Unzufriedenheit mit Engineering-Schnittstellen.
In: Computer & Automation, Heft 6/2012, S. 10-11.
- [HAREL & PNUELI 1985] **Harel, D., Pnueli, A.:**
On the Development of Reactive Systems.
In: Logics and Models of Concurrent Systems, NATO ASI Series, Band F-13, Springer-Verlag New York, 1985, S. 477-498.
- [HAREL 1987] **Harel, D.:**
Statecharts: a visual formalism for complex systems.
In: Science of Computer Programming 8, Heft 3/1987, S. 231-274.
- [HELLGREN ET AL. 2005] **Hellgren, A., Fabian, M., Lennartson, B.:**
On the execution of sequential function charts.
In: Control Engineering Practice 13, Heft 10/2005, S. 1283-1293.
- [HENZINGER ET AL. 1992] **Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.:**
Symbolic model checking for real-time systems.
In: Tagungsband "Seventh Annual IEEE Symposium on Logic in Computer Science", Ithaca (USA), 22.-25.06.1992, S. 394-406.
- [HIETTER ET AL. 2008] **Hietter, Y., Roussel, J.-M., Lesage, J.-J.:**
Algebraic synthesis of transition conditions of a state model.
In: Tagungsband "9th International Workshop on Discrete Event Systems", Göteborg (Schweden), 28.-30.05.2008, S. 187-192.
- [HILLAH ET AL. 2006] **Hillah, L. M., Kordon, F., Petrucci, L., Tréves, N.:**
PN standardisation: a survey.
In: Tagungsband "International Conference on Formal Methods for Networked and Distributed Systems", Paris (Frankreich), 26.-29.09.2006, S. 307-322.
- [HILLAH ET AL. 2009] **Hillah, L. M., Kindler, E., Kordon, F., Petrucci, L., Tréves, N.:**
A primer on the Petri Net Markup Language and ISO/IEC 15909-2.
In: Petri Net Newsletter 76, Oktober 2009, S. 9-28.
- [HILLAH & PETRUCCI 2010] **Hillah, L.M., Petrucci, L.:**
Standardisation des réseaux de Petri : état de l'art et enjeux futurs.
In: Génie Logiciel et Systemes Experts 93, 2010, S. 5-10.
- [HOFFSAES 1990] **Hoffsaes, C.:**
The French Society of Computer Scientists, AFCET.
In: Annals of the History of Computing 12, Heft 3/1990, S. 167-176.
- [HUUCK 2003] **Huuck, R.:**
Software Verification for Programmable Logic Controllers.
Dissertation an der Technischen Fakultät der Universität Kiel, Kiel, 2003.
- [JÄGER ET AL. 2012] **Jäger, T., Christiansen, L., Strube, M., Fay, A.:**
Durchgängige Werkzeugunterstützung von der Anforderungserhebung bis zur Anlagenstrukturbeschreibung mittels formalisierter Prozessbeschreibung und AutomationML.
In: Tagungsband "Fachtagung Entwurf komplexer Automatisierungssysteme", Magdeburg, 09.-10.05.2012, S. 239-252.
- [JENSEN 1997 A] **Jensen, K.:**
Coloured petri nets - Basic concepts, analysis methods and practical use. 2. Auflage, Band 1, Springer-Verlag Berlin u.a., 1997.
- [JENSEN 1997 B] **Jensen, K.:**
Coloured petri nets - Basic concepts, analysis methods and practical use. 2. Auflage, Band 2, Springer-Verlag Berlin u.a., 1997.

- [JENSEN 1997 C] **Jensen, K.:**
Coloured petri nets - Basic concepts, analysis methods and practical use. 2. Auflage, Band 3, Springer-Verlag Berlin u.a., 1997.
- [JOHN & TIEGELKAMP 2000] **John, K.-H.; Tiegelkamp, M.:**
SPS-Programmierung mit IEC 61131-3 – Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen. 3. Auflage, Springer Verlag Berlin u.a., 2000, S. 32-38.
- [JOHANSSON & ÅRZÉN 1999] **Johansson, C., Årzén, K. E.:**
Grafchart and Grafset: A comparison between two graphical languages aimed for sequential control applications.
In: Tagungsband "14th World Congress of IFAC", Peking (China), 05.-09.07.1999, S. 19-24.
- [KABRA ET AL. 2012] **Kabra, A., Bhattacharjee, A., Karmakar, G., Wakankar, A.:**
Formalization of Sequential Function Chart as Synchronous Model in LUSTRE.
In: Tagungsband "National Conference on Emerging Trends and Application in Computer Science", Mumbai (Indien), 30.-31.03.2012, S. 115-120.
- [KATZKE & VOGEL-HEUSER 2007] **Katzke, U., Vogel-Heuser, B.:**
Combining UML with IEC 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems.
In: Tagungsband "10th IFAC, IFIP, IFORS, IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems", Seoul (Korea), 04.-06.09.2007, S. 90-94.
- [KATZKE & VOGEL-HEUSER 2009] **Katzke, U., Vogel-Heuser, B.:**
Vergleich der Anwendbarkeit von UML und UML-PA in der anlagennahen Softwareentwicklung der Automatisierungstechnik.
In: Automatisierungstechnik - at 57, Heft 7/2009, S. 332-340.
- [KILLENBERG 1976] **Killenberg, H.:**
Verhaltensbeschreibung von Schaltsystemen mit Hilfe von Programmablaufgraphen.
In: „Messen, Steuern, Regeln“ 19, Heft 6/1976, S.197-201.
- [KLEIN ET AL. 2002] **Klein, S., Frey, G., Litz, L.:**
A Petri Net based Approach to the Development of correct Logic Controllers - Design, Verification, Validation, Evaluation and Implementation.
In: Tagungsband "2nd International Workshop on Integration of Specification Techniques for Applications in Engineering ", Grenoble (Frankreich), April 2002, S. 116-129.
- [KÖNIG & QUÄCK 1988] **König, R., Quäck, L.:**
Petri-Netze in der Steuerungs- und Digitaltechnik. Oldenbourg Verlag München Wien, 1988.
- [KOWALEWSKI 1995] **Kowalewski, S.:**
Modulare diskrete Modellierung verfahrenstechnischer Anlagen zum systematischen Steuerungsentwurf.
Dissertation am Lehrstuhl für Anlagensteuerungstechnik der Universität Dortmund, Dortmund, 1995.
- [KÜHNEL 2010] **Kühnel, A.:**
Visual C# 2010 - Das umfassende Handbuch. 5. Auflage, Galileo Press Bonn, 2010.

- [LE PARC ET AL. 1999] **Le Parc, P., L'Her, D., Scharbarg, J.-L., Marcé, L.:**
Grafcet revisited with a synchronous data-flow language.
In: IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans 29, Heft 3/1999, S. 284-293.
- [LESAGE & ROUSSEL 1993] **Lesage, J.-J., Roussel, J.-M.:**
Hierarchical Approach to GRAFCET using forcing order.
In: Automatique Productique Informatique Industrielle-APII 27, Heft 1/1993, S. 25-38.
- [LESAGE ET AL. 1996] **Lesage, J.-J., Roussel, J.-M., Thierry, C.:**
A Theory of Binary Signal.
In: Tagungsband "IEEE Multiconference on Computational Engineering in Systems Applications", Lille (Frankreich), 09.-12.07.1996, S. 590-595.
- [L'HER ET AL. 1995] **L'Her, D., Le Parc, P., Marcé, L.:**
Modeling and Proving Grafquets with Transition Systems.
In: Tagungsband "2nd AMAST workshop on Real-Time Systems and Project Models and Proofs", Bordeaux (Frankreich), Juni, 1995.
- [LHOSTE ET AL. 1993] **Lhoste, P., Panetto, H., Roesch, M.:**
GRAFCET: from syntax to semantics.
In: Automatique, productique, informatique industrielle 27, Heft 1/1993, S. 127-141.
- [LITZ & FREY 1999] **Litz, L., Frey, G.:**
Methoden und Werkzeuge zum industriellen Steuerungsentwurf, Historie, Stand, Ausblick.
In: Automatisierungstechnik - at 47, Heft 4/1999, S. 145-156.
- [LITZ 2005] **Litz, L.:**
Grundlagen der Automatisierungstechnik - Regelungssysteme, Steuerungssysteme, Hybride Systeme. Oldenbourg Wissenschaftsverlag München, 2005.
- [LJUNGKRANTZ & AKESSON 2007] **Ljungkrantz, O., Akesson, K.:**
A Study of Industrial Logic Control Programming using Library Components.
In: Tagungsband "IEEE Conference on Automation Science and Engineering", Scottsdale (USA), 22.-25.09.2007, S. 117-122.
- [LJUNGKRANTZ ET AL. 2010] **Ljungkrantz, O., Akesson, K., Martin, F., Chengyin, Y.:**
A formal specification language for PLC-based control logic.
In: Tagungsband "IEEE International Conference on Industrial Informatics", Osaka (Japan), 13.-16.07.2010, S. 1067-1072.
- [LOHMANN ET AL. 2007] **Lohmann, S., Thi, L.A.D., Tran, T.H., Engell, S., Stursberg, O.:**
Iterative Verfeinerung und Formalisierung von Spezifikationen im Kontext des systematischen Steuerungsentwurfs.
In: VDI-Berichte Nr. 1980, 2007, S. 3-12.
- [LOHMANN 2011] **Lohmann, S.:**
Systematic Logic Controller Design as Sequential Function Chart Starting from Informal Requirements.
Dissertation an der Fakultät für Bio- und Chemieingenieurwesen der Technischen Universität Dortmund, Dortmund, 2011.
- [LUCAS & TILBURY 2004] **Lucas, M. R., Tilbury, D. M.:**
The Practice of Industrial Logic Design.
In: Tagungsband "2004 American Control Conference", Boston (USA), 30.06.-02.07.2004, Band 2, S. 1350-1355.

- [LÜDER ET AL. 2010] **Lüder, A., Estévez, E., Hundt, L., Marcos, M.:**
Automatic transformation of logic control models within engineering of embedded mechatronical units.
In: International Journal of Advanced Manufacturing Technology 54, Heft 9-12/2010, S. 1077-1089.
- [LUNZE 2006] **Lunze, J.:**
Ereignisdiskrete Systeme - Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetzen. Oldenbourg Wissenschaftsverlag GmbH München, 2006.
- [MACHADO ET AL. 2001] **Machado, J.-M., Louni, F., Faure, J.-M., Lesage, J.-J., Ferreira da Silva, C.L., Roussel, J.-M.:**
Modeling and implementing the control of automated production systems using statecharts and PLC programming languages.
In: Tagungsband "European Control Conference", Porto (Portugal), 04.-07.09.2001.
- [MACHADO ET AL. 2006] **Machado, J., Denis, B., Lesage, J. J.:**
Formal Verification of Industrial Controllers: with or without a Plant model?.
In: Tagungsband "7th Portuguese Conference on Automatic Control", Lissabon (Portugal), 11.-13.09.2006, S. 341-346.
- [MACHADO & SEABRA 2010] **Machado, J., Seabra, E.:**
A systematized approach to obtain dependable controllers specifications.
In: ABCM Symposium Series in Mechatronics, Band 4, 2010, S. 408-417.
- [MACKIEWICZ 2006] **Mackiewicz, R. E.:**
Overview of IEC 61850 and Benefits.
In: Tagungsband "IEEE Power Engineering Society General Meeting", Montreal (Kanada), 18.-22.06.2006, S. 3195-3202.
- [MALLET ET AL. 2000] **Mallet, F., Gaffe, D., Boeri, F.:**
Concurrent control systems: from Grafcet to VHDL.
In: Tagungsband "26th EUROMICRO Conference", Maastricht (Niederlande), 05.-07.09.2000, S. 230-234.
- [MARCÉ ET AL. 1996] **Marcé, L., L'Her, D., Le Parc, P.:**
Modelling And Verification Of Temporized Grafcet.
In: Tagungsband "IEEE Multiconference on Computational Engineering in Systems Applications", Lille (Frankreich), 09.-12.07.1996, S. 783-788.
- [MARTIN 2011] **Martin, R.:**
Microsoft Visio 2010 Programmierung - Microsoft Visio effizient anpassen und erweitern. O'Reilly Verlag GmbH & Co. KG Köln, 2011.
- [MARTINEZ & MARTINEZ 2005] **Martínez, M. A., Martínez, J. L.:**
Specification of operations for a manipulator on a mobile robot using grafcet.
In: Robotica 23, Heft 6/2005, S. 789-791.
- [MERTKE & FREY 2001] **Mertke, T., Frey, G.:**
Formal Verification of PLC-programs generated from Signal Interpreted Petri Nets.
In: Tagungsband "IEEE International Conference on Systems, Man and Cybernetics", Tucson (USA), 07.-10.10.2001, S. 2700-2705.
- [MINAS & FREY 2002] **Minas, M., Frey, G.:**
Visual PLC-programming using signal interpreted Petri nets.
In: Tagungsband "American Control Conference", Anchorage (USA), 08.-10.05.2002, S. 5019-5024.

- [MOODY 2009] **Moody, D. L.:**
The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering.
In: IEEE Transactions on Software Engineering 35, Heft 6/2009, S. 756-779.
- [NAITO & TSUNOYAMA 1981] **Naito, S. Tsunoyama, M.:**
Fault Detection for Sequential Machines by Transitions Tours.
In: Tagungsband "IEEE Fault Tolerant Computer Symposium, Portland (USA), 24.-26.06.1981, S. 238-243.
- [NKE & LUNZE 2013] **Nke, Y., Lunze, J.:**
Systematischer Entwurf fehlertoleranter Steuerungen.
In: Automatisierungstechnik - at 61, Heft 2/2013, S. 122-130.
- [PAL 2007] **Prüfungsaufgaben- und Lehrmittelentwicklungsstelle IHK Region Stuttgart (Hrsg.):**
Informationen für die Praxis - Information zur DIN EN 60848, Erklärung und Anwendung der DIN EN 60848 „GRAFCET, Spezifikationsprache für Funktionspläne der Ablaufsteuerung“ in den PAL-Prüfungen, April 2007.
- [PETRI 1962] **Petri, C. A.:**
Kommunikation mit Automaten.
Dissertation an der Fakultät für Mathematik und Physik der Technischen Hochschule Darmstadt, Bonn, 1962.
- [PHILIPPOT & TAJER 2010] **Philippot, A., Tajer, A.:**
From GRAFCET to Equivalent Graph for Synthesis Control of Discrete Events Systems. In: Tagungsband "18th Mediterranean Conference on Control & Automation", Marrakech (Marokko), 23.-25.06.2010, S. 683-688.
- [PIEDRAFITA & VILLARROEL 2008] **Piedrafita, R., Villarroel, J. L.:**
Evaluation of Sequential Function Charts execution techniques. The Active Steps Algorithm.
In: Tagungsband "IEEE International Conference on Emerging Trends and Factory Automation", Hamburg, 15.-18.09.2008, S. 74-81.
- [PONSA ET AL. 2009] **Ponsa, P., Vilanova, R., Amante, B.:**
Towards integral human-machine system conception: From automation design to usability concerns.
In: Tagungsband "IEEE International Conference on Human System Interactions", Catania (Italien), 21.-23.05.2009, S. 427-433.
- [PROVOST ET AL. 2009] **Provost, J., Roussel, J.-M., Faure, J.-M.:**
Test sequence construction from SFC specification.
In: Tagungsband "2nd IFAC Workshop on Dependable Control of Discrete Systems", Bari (Italien), 10.-12.06.2009, S. 299-304.
- [PROVOST ET AL. 2010 A] **Provost, J., Roussel, J.-M., Faure, J.-M.:**
Translating Grafcet specifications into Mealy machines for conformance test purposes. In: Control Engineering Practice 19, Heft 9/2011, S. 947-957.
- [PROVOST ET AL. 2010 B] **Provost, J., Roussel, J.-M., Faure, J.-M.:**
SIC-testability of sequential logic controllers.
In: Tagungsband "10th International Workshop on Discrete Event Systems", Berlin, 30.08.-01.09.2010, S. 193-198.
- [PROVOST ET AL. 2011 A] **Provost, J., Roussel, J.-M., Faure, J.-M.:**
Testing programmable logic controllers from finite state machines specification.
In: Tagungsband "3rd International Workshop on Dependable Control of Discrete Systems", Saarbrücken, 15.-17.06.2011, S. 1-6.

- [PROVOST ET AL. 2011 B] **Provost, J., Roussel, J.-M., Faure, J.-M.:**
A formal semantics for Grafset specifications.
In: Tagungsband "IEEE International Conference on Automation Science and Engineering", Triest (Italien), 24.-27.08.2011, S. 488-494.
- [RAMADGE & WONHAM 1986] **Ramadge, P. J., Wonham, W. M.:**
Supervisory control of a class of discrete event processes.
In: Tagungsband "Sixth International Conference on Analysis and optimization of Systems", Nizza (Italien), 19.-22.06.1984, S. 475-498.
- [RAMCHANDANI 1973] **Ramchandani, C.:**
Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.
Dissertation am Massachusetts Institute of Technology, Cambridge (USA), 1974.
- [RAUSCH & HANISCH 1995] **Rausch, M., Hanisch, H.-M.:**
Net condition/event systems with multiple condition outputs.
In: Tagungsband "INRIA/IEEE Symposium on Emerging Technologies and Factory Automation", Paris (Frankreich), 10.-13.10.2013, S. 592-600.
- [ROUSSEL 1994] **Roussel, J.-M.:**
Analyse de Grafsets par Génération Logique de l'Automate Équivalent.
Dissertation an der École Normale Supérieure de Cachan, Cachan (Frankreich), 1994.
- [ROUSSEL & LESAGE 1996] **Roussel, J.-M., Lesage, J.-J.:**
Validation and verification of GRAFCETs using finite state machine.
In Tagungsband "IEEE Multiconference on Computational Engineering in Systems Applications", Lille (Frankreich), 09.-12.07.1996, S. 758-764.
- [ROUSSEL ET AL. 1999] **Roussel, J.-M., Lamperiere-Couffin, S., Lesage, J.-J.:**
IEC 60848 et IEC 61131-3: deux normes complémentaires. Journée d'études "Nouvelles percées dans les langages pour l'automatique". Publikation der Société de l'Electricité, de l'Electronique et des TIC, "SEE-Club 18 (Automatique et Automatisation Industrielle)", Amiens (Frankreich), 25.11.1999.
- [ROUSSEL & FAURE 2002] **Roussel, J.-M., Faure, J.-M.:**
An algebraic approach for PLC programs verification.
In: Tagungsband "6th International Workshop on Discrete Event Systems", Saragossa (Spanien), 02.-04.10.2002, S. 303-310 .
- [ROUSSEL ET AL. 2004] **Roussel, J.-M., Faure, J.-M., Lesage, J.-J., Medina, A.:**
Algebraic approach for dependable logic control systems design.
In: International Journal of Production Research 42, Heft 14/2004, S. 2859-2876.
- [SARMENTO ET AL. 2008] **Sarmiento, C. A., Silva, J. R., Miyagi, P. E., Santos Filho, D. J.:**
Modeling of Programs and its Verification for Programmable Logic Controllers.
In: Tagungsband "17th IFAC World Congress", Seoul (Korea), 06.-11.07.2008, S. 10546-10551.
- [SCHNIEDER 1999] **Schnieder, E.:**
Methoden der Automatisierung. Beschreibungsmittel, Modellkonzepte und Werkzeuge für Automatisierungssysteme mit 56 Tabellen. Vieweg Verlag Braunschweig, 1999.
- [SEITZ 2003] **Seitz, M.:**
Speicherprogrammierbare Steuerungen, Von den Grundlagen der Prozessautomatisierung bis zur vertikalen Integration. Fachbuchverlag Leipzig im Carl Hanser Verlag München Wien, 2003, S. 19-20.

- [SELIC 2003] **Selic, B.:**
The Pragmatics of Model-Driven Development.
In: IEEE Software 20, Heft 5/2003, S. 19-25.
- [SREENIVAS & KROGH 1991] **Sreenivas, R.S., Krogh, B.H.:**
On Condition/Event Systems with Discrete State Realizations.
In: Discrete Event Dynamic Systems: Theory and Application 1, Heft 2/1991, S. 209-236.
- [STURSBURG 2012] **Stursberg, O.:**
Hierarchical and Distributed Discrete Event Control of Manufacturing Processes.
In: Tagungsband "IEEE Conference on Emerging Technology and Factory Automation", Krakau (Polen), 17.-21.09.2012.
- [TAUCHNITZ 2013] **Tauchnitz, T.:**
Integriertes Engineering - wann, wenn icht jetzt! - Notwendigkeit, Anforderungen und Ansätze.
In: Automatisierungstechnische Praxis - atp 55, Heft 1-2/2013, S. 46-53.
- [THIEME & HANISCH 2002] **Thieme, J., Hanisch, H.-M.:**
Model-based generation of modular PLC code using IEC61131 function blocks.
In: Tagungsband "IEEE International Symposium on Industrial Electronics", L'Aquila (Italien), 08.-11.07.2002, S. 199-204.
- [THOMAS & MCLEAN 1988] **Thomas, B.H., McLean, C.:**
Using Grafcet to design generic controllers.
In: Tagungsband "International Conference on Computer Integrated Manufacturing", Troy (USA), 23.-25.05.1988, S. 110-119.
- [ULRICH ET AL. 2009] **Ulrich, A., Güttel, K., Fay, A.:**
Durchgängige Prozesssicht in unterschiedlichen Domänen - Methoden und Werkzeug zum Einsatz der formalisierten Prozessbeschreibung.
In: Automatisierungstechnik - at 57, Heft 2/2009, S. 80-92.
- [VIDANAPATHIRANA ET AL. 2011] **Vidanapathirana, A. C., Dewasurendra, S. D., Abeyratne, S.G.:**
Statechart Based Modeling and Controller Implementation of Complex Reactive Systems.
In: Tagungsband "6th International Conference on Industrial and Information Systems", Kandy (Sri Lanka), 16.-19.08.2011, S. 493-498.
- [VOGEL-HEUSER ET AL. 2013] **Vogel-Heuser, B., Diedrich, C., Fay, A., Göhner, P.:**
Anforderungen an das Software-Engineering in der Automatisierungstechnik.
In: Tagungsband „Software Engineering (SE 13)“, Aachen, 26.02.-01.03.2013.
- [VON DER BEECK 1994] **von der Beeck, M.:**
A Comparison of Statecharts Variants.
In: Tagungsband "Third International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems", Lübeck, 19.-23.09.1994, S. 128-148.
- [WAGNER ET AL. 2007] **Wagner, F., Münch, P., Liu, S., Frey, G.:**
Development Process for dependable high-performance controllers using Petri Nets and FPGA Technology.
In: Tagungsband "IFAC Workshop on Dependable Control of Discrete Systems", Cachan (Frankreich), 13.-15.06.2007, S. 297-302.
- [WALLÉN 1995] **Wallén, A.:**
Using Grafcet To Structure Control Algorithms.
In: Tagungsband "Third European Control Conference", Rom (Italien), 05.-08.09.1995, S. 3160-3165.

- [WIGHTKIN ET AL. 2011] **Wightkin, N., Buy, U., Darabi, H.:**
Formal Modeling of Sequential Function Charts With Time Petri Nets.
In: IEEE Transactions on Control Systems Technology 19, Heft 2/2011, S. 455-464.
- [WITSCH ET AL. 2008] **Witsch, D., Wannagat, A., Vogel-Heuser, B.:**
Entwurf wiederverwendbarer Steuerungssoftware mit Objektorientierung und UML.
In: Automatisierungstechnische Praxis - atp 50, Heft 12/2008, S. 54-60.
- [WITSCH & VOGEL-HEUSER 2009] **Witsch, D., Vogel-Heuser, B.:**
Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions.
In: Tagungsband "IEEE International Conference on Emerging Trends and Factory Automation", Mallorca (Spanien), 22.-26.09.2009, doi: 10.1109/ETFA.2009.5347155.
- [WITSCH ET AL. 2010] **Witsch, D., Ricken, M., Kormann, B., Vogel-Heuser, B.:**
PLC-Statecharts: An Approach to Integrate UML-Statecharts in Open-Loop Control Engineering.
In: Tagungsband "8th IEEE International Conference on Industrial Informatics", Osaka (Japan), 13.-16.07.2010, S. 915-920.
- [WOLLSCHLAEGER & WENZEL 2005] **Wollschlaeger, M., Wenzel, P.:**
Common model and infrastructure for application of XML within the automation domain.
In: Tagungsband "IEEE International Conference on Industrial Informatics", Perth (Australien), 10.-12.08.2005, S. 246-251.
- [WOLLSCHLAEGER ET AL. 2010] **Wollschlaeger, M., Mühlhause, M., Runde, S., Lindemann, L.:**
XML in der Automation – Systematisches Sprachdesign.
In: Tagungsband "Fachtagung, Entwurf komplexer Automatisierungssysteme", Magdeburg, 25.-27.05.2010, S. 19-27.
- [ZAYTOON & CARRÉ-MÉNÉTRIER 2001] **Zaytoon, J., Carré-Ménétrier, V.:**
Synthesis of control implementation for discrete manufacturing systems.
In: International Journal of Production Research 39, Heft 2/2001, S. 329-345.
- [ZAYTOON 2002] **Zaytoon, J.:**
On the recent advances in Grafcet.
In: Production Planning & Control 13, Heft 3/2002, S. 85-100.

Referenzierte Normen, Richtlinien und Empfehlungen

Dieses Verzeichnis enthält referenzierte Normen, Richtlinien und Empfehlungen.

Auf sie wird mit [<KURZBELEG>][#] verwiesen.

- [ISO 15745-1][#] Industrial automation systems and integration - Open systems application integration framework - Part 1: Generic reference description. (2003)
- [ISO 15745-3][#] Industrial automation systems and integration - Open systems application integration framework - Part 3: Reference description for IEC 61158-based control systems. (2003)
- [ISO/IEC 15909-1][#] Software and system engineering - High-level Petri nets - Part 1: Concepts, definitions and graphical notation. (2004)
- [ISO/IEC 15909-2][#] Software and system engineering - High-level Petri nets - Part 2: Transfer format. (2011)
- [ISO/IEC 19757-2][#] Information technology - Document Schema Definition Language (DSDL) - Part 2: Regular-grammar-based validation - RELAX NG. (2008)
- [IEC 848 A][#] GRAFCET - specification language for sequential function charts. (1988)
- [IEC 848 B][#] GRAFCET - specification language for sequential function charts. (1992)
- [IEC 60848 A][#] GRAFCET - specification language for sequential function charts. (2002)
- [IEC 60848 B][#] GRAFCET - specification language for sequential function charts. (2013)
- [IEC 61131-3][#] Programmable Controllers - Part 3: Programming Languages. (2013)
- [IEC 61499-1][#] Function blocks – Part 1: Architecture. (2013)
- [ISA 88.01][#] /
[IEC 61512-2][#] Batch Control - Part 2: Data Structures and Guidelines for Languages. (2001)
- [IEC 62264-1][#] IEC/DIS 62264-1: Enterprise-control system integration - Part 1: Models and terminology. (2012)
- [IEC 62424][#] Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools. (2008)
- [65E/280/CDV][#] 65E/280/CDV Committee Draft for Voting (IEC 62714-1): Engineering data exchange format for use in industrial automation systems engineering- AutomationML- Part 1 - architecture and general requirement. (2013)
- [DIN EN 60848][#] GRAFCET - Spezifikationsprache für Funktionspläne der Ablaufsteuerung. (2002)
- [DIN EN 61131-1][#] Speicherprogrammierbare Steuerungen - Teil 1: Allgemeine Informationen. (2003)
- [DIN EN 61131-3][#] Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen. (2003)
- [DIN EN 61131-3, BEIBLATT 1][#] Speicherprogrammierbare Steuerungen - Leitlinien für die Anwendung und Implementierung von Programmiersprachen für Speicherprogrammierbare Steuerungen. (2005)
- [DIN 40719-6][#] Schaltungsunterlagen – Teil 6: Regeln für Funktionspläne (IEC 60848:1988 modifiziert). (1992)
- [NF C 03-190][#] Diagramme fonctionnel GRAFCET pour la description des systèmes logiques de commande. (1982)
- [ADEPA 1981][#] GEMMA (Guide d'Étude des Modes de Marches et d'Arrêts). Agence nationale pour le Développement de la Production Automatisée. (1981)

- [IEEE 1076][#] IEEE Standard 1076 - VHDL Language Reference Manual. (1987)
- [OMG UML, v2.4.1][#] Object Management Group - Unified Modeling Language: Infrastructure, Version 2.4.1. (2011)
- [PLCOPEN 2009][#] Technical Paper, PLCopen Technical Committee 6: XML Formats for IEC 61131-3, Version 2.01 - Official Release. (2009)
- [PROFINET 2006][#] Profiles for distributed automation-Specification for PROFINET (Version 2.2). (2006)
- [W3C XML 1.0][#] Extensible Markup Language (XML) 1.0 (Fifth Edition) - W3C Recommendation. (2008)
- [W3C XSD 1.1 PART 1][#] XML Schema Definition Language (XSD) 1.1 Part 1: Structures - W3C Recommendation (2012)
- [W3C XSD 1.1 PART 2][#] XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes - W3C Recommendation (2012)
- [W3C XSLT 2.0][#] XSL Transformations (XSLT) Version 2.0 - W3C Recommendation (2007)
- [VDI/VDE 3682][#] VDI/VDE-Richtlinie 3682: Formalisierte Prozessbeschreibungen. (2005)
- [VDI/VDE 3690, BLATT 1][#] VDI/VDE-Richtlinie 3690-1: XML in der Automation - Klassifikation ausgewählter Anwendungen. (2011)
- [VDI/VDE 3690, BLATT 2][#] VDI/VDE-Richtlinie 3690-2: XML in der Automation - Überführung fachlicher Modelle nach XML. (2012)
- [VDI/VDE 3694][#] VDI/VDE-Richtlinie 3694: Lastenheft / Pflichtenheft für den Einsatz von Automatisierungssystemen. (2008)

Referenzierte Internetquellen und Software

Dieses Verzeichnis enthält referenzierte Internetquellen und Software.

Entsprechende Verweise werden durch [KURZBELEG]@ kenntlich gemacht.

- [AUTOMATION 2012][@] VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (Hrsg.): *Umfrageergebnisse zum Kongress Automation 2012*. - <http://www.vdi.de/artikel/deutsche-automatisierer-haben-komplexe-aufgaben-im-griff/> [letzter Zugriff: 19.02.2013]
- [CODESYS][@] 3S-Smart Software Solutions GmbH: *CODESYS* - <http://www.codesys.com/> [letzter Zugriff: 03.01.2013]
- [COLLADA][@] *COLLADA* - <http://www.khronos.org/collada/> [letzter Zugriff: 22.01.2013]
- [CONTROL BUILD][@] Dassault Systemes: *Control Build* - <http://www.3ds.com/de/products/catia/portfolio/geensoft/controlbuild/> [letzter Zugriff: 25.03.2013]
- [CONTROL BUILDER][@] ABB: *Control Builder AC500* - <http://www.abb.de/product/ap/seitp329/a0ca45803dc11730c1257013002edb64.aspx> [letzter Zugriff: 25.03.2013]
- [EPNK][@] Eclipse-based Petri Net Kernel (*ePNK*) - <http://www2.imm.dtu.dk/~ekki/projects/ePNK/> [letzter Zugriff: 07.02.2013]
- [FLUIDSIM][@] Festo Didactic: *FluidSim* - <http://www.festo-didactic.com/de-de/lernsysteme/software-e-learning/fluidsim/> [letzter Zugriff: 02.01.2013]
- [GMA][@] Gesellschaft Mess- und Automatisierungstechnik im VDI/VDE (*GMA*) - <http://www.vdi.de/technik/fachthemen/mess-und-automatisierungstechnik/> [letzter Zugriff: 21.01.2013]
- [JASPERNEITE 2012][@] Jasperneite, J.: *Internet und Automation - Was hinter den Begriffen wie Industrie 4.0 steckt*. - http://www.computer-automation.de/steuerungsebene/steuernregeln/fachwissen/article/93559/1/Was_hinter_Begriffen_wie_Industrie_40_steckt/ [letzter Zugriff: 12.02.2013]
- [JAVA][@] Oracle: *Java* - <http://www.java.com/de/> [letzter Zugriff: 02.01.2013]
- [JGRAFCHART][@] *JGrafchart* - <http://www3.control.lth.se/grafchart/download/download.pike> [letzter Zugriff: 02.01.2013]
- [LINQ][@] *LINQ to XML* - <http://msdn.microsoft.com/de-de/library/vstudio/bb387098.aspx> [letzter Zugriff: 06.02.2013]
- [LOGI.CALS][@] logi.cals automation solutions & services GmbH: *Logi.CAD* - <http://www.logicals.com/products/logi.CAD/> [letzter Zugriff: 14.05.2013]
- [LP-RHEINLAND-PFALZ 2011][@] Ministerium für Bildung, Wissenschaft Weiterbildung und Kultur (Hrsg.): *Lehrplan für das berufliche Gymnasium, Unterrichtsfach Technik, Schwerpunkt Elektrotechnik* - http://bbs.bildung-rp.de/fileadmin/user_upload/bbs/bbs.bildung-rp.de/materialien/lehrplaene/Lehrplan_2010_11/BG_Lehrplan_KomplettTechnik_ET.pdf [letzter Zugriff: 16.12.2012].
- [LUND][@] Department of Automatic Control, *LUND University* - <http://www.control.lth.se/> [letzter Zugriff: 02.01.2013]

- [LURPA][@] Laboratoire Universitaire de Recherche en Production Automatisée (*LURPA*), Groupe GRAFCET - http://www.lurpa.ens-cachan.fr/grafcet/grafcet_fr.html [letzter Zugriff: 17.12.2012]
- [LUSTRE][@] *Lustre* - http://wiki.lustre.org/index.php/Main_Page [letzter Zugriff: 07.01.2013]
- [MATLAB/SIMULINK][@] MathWorks: *MATLAB/Simulink* - <http://www.mathworks.de/products/simulink/> [letzter Zugriff: 05.03.2013]
- [MICROSOFT][@] *Microsoft Office 2010, Microsoft Excel 2010, Microsoft Visio 2010* - <http://office.microsoft.com> [letzter Zugriff: 04.02.2013]
- [MULTIPROG][@] KW-Software GmbH: *Multiprog* - <https://www.kw-software.com/de/iec-61131-control/programmiersysteme/multiprog-pro> [letzter Zugriff: 04.01.2013]
- [PCWORX][@] Phoenix Contact Deutschland GmbH: *PC WORX* - https://www.phoenixcontact.com/online/portal/de?1dmy&urile=wcm%3apath%3a/dede/web/main/products/subcategory_pages/programming_p-19-05/8b777145-e7f2-4eaa-ae5e-4dadce30223/8b777145-e7f2-4eaa-ae5e-4dadce30223 [letzter Zugriff: 01.05.2013]
- [PLCOPEN][@] *PLCopen* - <http://www.plcopen.org/> [letzter Zugriff: 04.01.2013]
- [PROFIBUS / PROFINET][@] *PROFIBUS & PROFINET Nutzerorganisation e.V.* - <http://www.profibus.com/> [letzter Zugriff: 22.01.2013]
- [SFCEDIT][@] *SFCedit* - <http://stephane.dimeglio.free.fr/sfcedit/en/> [letzter Zugriff: 02.01.2013]
- [STATEFLOW][@] MathWorks: *Stateflow* - <http://www.mathworks.de/products/stateflow/> [letzter Zugriff: 05.03.2013]
- [STEP 7][@] Siemens AG: *STEP 7* - <http://www.automation.siemens.com/mcms/simatic-controller-software/de/step7/seiten/default.aspx> [letzter Zugriff: 04.01.2013]
- [TINA][@] Time Petri Net Analyzer (*TINA*) - <http://projects.laas.fr/tina/> [letzter Zugriff: 07.02.2013]
- [TS-HANNOVER 2010][@] Technikerschule Hannover, Fachschule Maschinentechnik (Hrsg.): *Lehrplan für die Fachschule Maschinentechnik, FSM-Curriculum 2010* - http://www.bbs-me.de/fileadmin/material/technikerschule/FSM/FS-LPL-03-REV_04__FSM-Curriculum_Maschinentechnik_2010.pdf [letzter Zugriff: 16.12.2012].
- [TWINCAT][@] Beckhoff Automation GmbH: *TwinCAT* - <http://www.beckhoff.de/default.asp?twincat/default.htm> [letzter Zugriff: 01.05.2013]
- [VDMA 2012][@] Verband Deutscher Maschinen- und Anlagenbau e.V. (Hrsg.): *Trendstudie: IT und Automation in den Produkten des Maschinenbaus bis 2015 (Kurzzusammenfassung)*. Verfügbar unter: <http://www.vdma.org/downloads> [letzter Zugriff: 19.02.2013]
- [VISIO][@] *Visio Corporation* - <http://visio.mvps.org/History/default.html> [letzter Zugriff: 04.02.2013]
- [W3C][@] World Wide Web Consortium (*W3C*) - <http://www.w3.org/Consortium/mission.html> [letzter Zugriff: 21.01.2013]
- [WINERS][@] *WinErs GRAFCET* - http://www.schoop-didactic.de/index.php?option=com_content&view=category&id=94&Itemid=115 [letzter Zugriff: 02.01.2013]

Veröffentlichungen des Verfassers

Dieses Verzeichnis enthält eine Liste der Veröffentlichungen des Verfassers.
Sie werden durch [<KURZBELEG>]* referenziert.

- [CHRISTIANSEN ET AL. 2012]* **Christiansen, L., Jäger, T., Schumacher, F., Fay, A.:**
Modellierungsvorschlag zur grafischen Beschreibung alternativer und paralleler Prozessabläufe auf Basis eines Vergleichs bestehender Beschreibungsmittel.
In: Tagungsband "Fachtagung Entwurf komplexer Automatisierungssysteme", Magdeburg, 09.-10.05.2012, S. 1-12.
- [SCHUMACHER & FAY 2011]* **Schumacher, F., Fay, A.:**
Requirements and obstacles for the transformation of GRAFCET specifications into IEC 61131-3 PLC programs.
In: Tagungsband "IEEE International Conference on Emerging Trends and Factory Automation", Toulouse (Frankreich), 05.-09.09.2011.
- [SCHUMACHER & FAY 2013 A]* **Schumacher, F., Fay, A.:**
Transforming time constraints of a GRAFCET graph into a suitable Petri net formalism.
In: Tagungsband "IEEE International Conference on Industrial Technology", Kapstadt (Südafrika), 25.-27.02.2013, S. 210-218.
- [SCHUMACHER & FAY 2013 B]* **Schumacher, F., Fay, A.:**
Konzept und Werkzeugunterstützung zur automatischen Generierung von IEC 61131-3 konformen Steuerungsalgorithmen auf Basis einer GRAFCET-Spezifikation.
In: Tagungsband "Automationskongress 2013", Baden-Baden, 25.-26.06.2013.
- [SCHUMACHER ET AL. 2013 A]* **Schumacher, F., Schröck, S., Fay, A.:**
Transforming hierarchical concepts of GRAFCET into a suitable Petri net formalism. In: Tagungsband "IFAC Conference on Manufacturing Modelling, Management and Control", St. Petersburg (Russische Föderation), 19.-21.06.2013.
- [SCHUMACHER ET AL. 2013 B]* **Schumacher, F., Wolf, G., Drumm, O., Fay, A.:**
Anforderungen an die Feldgerätesimulation im Lebenszyklus von Anlagen.
In: Tagungsband "Automationskongress 2013", Baden-Baden, 25.-26.06.2013.

Studentische Arbeiten

Dieses Verzeichnis enthält eine Liste der unter Anleitung des Verfassers durchgeführten studentischen Arbeiten am Institut für Automatisierungstechnik der Helmut-Schmidt-Universität/ Universität der Bundeswehr Hamburg. Studienarbeiten werden durch [<KURZBELEG_SA>], Masterarbeiten durch [<KURZBELEG_MA>] gekennzeichnet.

- [DZIEDO 2012_SA] **Dziedo, A.:**
Anwendung von GRAFCET zur Spezifikation funktionaler Anforderungen im Engineering abwassertechnischer Anlagen. Studienarbeit, 2012.
- [DZIEDO 2012_MA] **Dziedo, A.:**
Anwendung einer Komplexitätsmetrik für steuerungstechnisch interpretierte Petrinetze auf GRAFCET nach IEC 60848. Masterarbeit, 2012.

- [GEBHARDT
2011_MA] **Gebhardt, C.:**
*Entwicklung eines softwarebasierten Werkzeugs für die automatische
Generierung von Steuerungscode auf Basis der Formalisierten
Prozessbeschreibung nach VDI/VDE-Richtlinie 3682. Masterarbeit, 2011.*
- [PINGEL 2012_SA] **Pingel, C.:**
*Konzept für die Abbildung werkzeugspezifischer GRAFCET-Spezifikationsdaten
in die Petri Net Markup Language nach ISO 15909-2. Studienarbeit, 2012.*
- [PINGEL 2012_MA] **Pingel, C.:**
*Entwicklung eines Softwarewerkzeugs für die Abbildung werkzeug-spezifischer
GRAFCET-Spezifikationsdaten in die Petri Net Markup Language nach ISO
15909-2. Masterarbeit. 2012.*
- [SCHOLZ 2012_SA] **Scholz, S.:**
*Transformationskonzept für die Abbildung einer GRAFCET-Spezifikation in
eine Implementierung gemäß der IEC 61131-3. Studienarbeit, 2012.*
- [SCHOLZ 2012_MA] **Scholz, S.:**
*Entwicklung eines XSLT-Stylesheets zur software-gestützten Trans-formation
von GRAFCET in eine Implementierung nach IEC 61131-3. Masterarbeit, 2012.*
- [SCHRÖCK
2012_MA] **Schröck, S.:**
*Entwicklung eines softwarebasierten Werkzeugs als Grundlage für die
automatische Generierung von Steuerungscode auf Basis einer
Steuerungsspezifikation mit GRAFCET (IEC 60848). Masterarbeit, 2012.*

Lebenslauf

Persönliche Daten

Name: Frank Schumacher
Geburtsdaten: 17.04.1983 in Siegen
Familienstand: verheiratet

Schulbildung

08/1989 – 06/1993 Grundschule in Alchen
08/1993 - 06/2002 Fürst-Johann-Moritz Gymnasium in Siegen
Abschluss: Abitur

Studium

10/2003 – 04/2007 Studium der Rechnergestützten Ingenieurwissenschaften mit Vertiefung in „Automatisierungs- und Informationstechnik“ an der Helmut-Schmidt-Universität/Universität der Bundeswehr Hamburg
Abschluss: Dipl.-Ing.

Beruf

07/2002 – 09/2002 Grundwehrdienst beim 6./ Luftwaffenausbildungsregiment 3, Bayreuth
10/2002 – 09/2003 Offizierausbildung im Truppendienst der Luftwaffe
Dienstteilbereich: Flugabwehrraketendienst
04/2007 – 12/2010 Einsatzoffizier bei der 3./ Flugabwehrraketenstaffel 21, Sanitz
seit 01/2011 Wissenschaftlicher Mitarbeiter
an der Professur für Automatisierungstechnik
der Helmut-Schmidt-Universität/Universität der Bundeswehr Hamburg