Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1254

Dipl.-Ing. Andreas Schüller,
Aachen

Ein Referenzmodell zur Beschreibung allgemeiner Prozeduren im leittechnischen Umfeld

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Ein Referenzmodell zur Beschreibung allgemeiner Prozeduren im leittechnischen Umfeld

Von der Fakultät für Georessourcen und Materialtechnik der
Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

vorgelegt von

Dipl.-Ing. Andreas Heinrich Schüller

aus Aachen

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ.-Prof. Dr.-Ing. Alexander Fay

Tag der mündlichen Prüfung: 09.11.2016

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

Dipl.-Ing. Andreas Schüller, Aachen

Nr. 1254

Ein Referenzmodell zur Beschreibung allgemeiner Prozeduren im leittechnischen Umfeld

Lehrstuhl für Prozessleittechnik der RWTH Aachen

Schüller, Andreas
Ein Referenzmodell zur Beschreibung allgemeiner Prozeduren im leittechnischen Umfeld
Fortschr.-Ber. VDI Reihe 8 Nr. 1254. Düsseldorf: VDI Verlag 2016.
162 Seiten, 68 Bilder, 19 Tabellen.
ISBN 978-3-18-525308-6, ISSN 0178-9546,
€ 62,00/VDI-Mitgliederpreis € 55,80.

Für die Dokumentation: Prozessleittechnik – Prozedur – Modellierung – Ablaufbeschreibung – Referenzmodell – Beschreibungssprache – Dienste – Interaktion

Die vorliegende Arbeit wendet sich an Ingenieure und Wissenschaftler im Bereich der Automatisierungstechnik. Sie befasst sich mit der Erstellung eines Referenzmodells zur Prozedurbeschreibung. Das Modell baut auf einer umfangreichen Analyse verschiedener bereits existierender Prozedurbeschreibungssprachen auf. Das Referenzmodell bildet eine Abstraktionsebene, die den gemeinsamen Kern der verschiedenen Beschreibungssprachen enthält. Diese Abstraktionsebene ermöglicht eine semantische Verknüpfung von Steuerungsprozeduren für technische Prozesse und Geschäftsprozesse. Die Funktionen des Steuernden und die des Ausführenden können im Referenzmodell sowohl durch Menschen als auch durch Maschinen übernommen werden. Umgesetzt wird dies durch die Verwendung eines Dienstsystems, das eine strikte Trennung zwischen den steuernden und den ausführenden Akteuren ermöglicht.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter http://dnb.ddb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at http://dnb.ddb.de.

D 82 (Diss. RWTH Aachen University, 2016)

ISSN 0178-9546
ISBN 978-3-18-525408-6

Vorwort

Diese Dissertation habe ich während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Prozessleittechnik an der RWTH Aachen University erstellt. Während dieser Zeit konnte ich auf eine große Unterstützung durch das gesamte Team des Lehrstuhls bauen, für die ich mich an dieser Stelle bedanken möchte.

Zunächst gilt mein besonderer Dank Herrn Professor Dr.-Ing. Ulrich Epple. Die angenehme Arbeitsatmosphäre an seinem Lehrstuhl, viele intensive fachliche Diskussionen mit ihm und persönliche Gespräche waren die Basis zum Gelingen dieser Arbeit. Durch die vielfältigen Aufgaben am Lehrstuhl und die Möglichkeit der Teilnahme an Fachausschüssen und Arbeitskreisen hat er es mir ermöglicht den Blick auch über den Tellerrand meines Dissertationsthemas hinweg richten zu können.

Herrn Professor Dr.-Ing. Alexander Fay, Inhaber der Professur für Automatisierungstechnik an der Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg, danke ich für die Übernahme der Zweitbegutachtung. Seine konstruktiven Kommentare haben mir eine weitere Sicht auf das Thema gegeben. Bei Herrn Univ.-Prof. Dr.-Ing. Karl Nienhaus bedanke ich mich für die Übernahme des Prüfungsvorsitzes.

Ein großer Dank gilt auch meinen Kollegen am Lehrstuhl. Die intensiven Diskussionen mit ihnen haben mir oft geholfen, neue Lösungswege zu erkennen. Erwähnen möchte ich an dieser Stelle Herrn Dipl.-Inform. Markus Schlütter, der mich als langjähriger Betreuer während meiner Zeit als studentische Hilfskraft an das Thema Prozessleittechnik herangeführt hat. Ein Dank gilt auch meinen ehemaligen studentischen Hilfskräften und den Studierenden, die bei mir ihre Abschlussarbeit geschrieben haben.

Bedanken möchte ich mich auch bei den Mitgliedern des NAMUR Arbeitskreises 1.11 „Referenzmodelle der Leittechnik“. Im Rahmen des Arbeitskreises durfte ich meine Arbeiten zur Prozedurbeschreibung vorstellen. Die konstruktiven Diskussionen haben dafür gesorgt, dass meine Arbeit einen Bezug zur Praxis erhalten hat.

Herzlich bedanken möchte ich mich auch bei Frau Ursula Bey und bei Frau Margarete Milescu-Huber, die mir bei der Bewältigung der Hürden in der Hochschulbürokratie geholfen haben und mir während der Zeit am Lehrstuhl ebenso wie Frau Martina Uecker jederzeit mit Rat und Tat zur Seite standen. Für die abschließende Durchsicht möchte ich mich besonders bei Frau Bey und bei Frau Simone Claßen bedanken.

Auch meiner Verlobten Frau Dr.-Ing. Nadin Schlegel möchte ich ein großes Danke schön für ihre Unterstützung und ihre Geduld sagen. Schließlich möchte ich mich bei meiner Familie bedanken, insbesondere bei meinen Eltern Mathias und Anita Schüller, die mich zu diesem beruflichen und privaten Weg geführt haben. Danke!

Aachen, im November 2016 *Andreas Schüller*

Nichts beflügelt die Wissenschaft so wie der Schwatz mit Kollegen auf dem Flur.
Arno Penzias (*1933), amerik. Physiker.

Inhaltsverzeichnis

Akronyme

ABK	Anzeige-/Bedienkomponente
AD	Aktivitätsdiagramme
AP	Automatisierungspyramide
ARIS	Architektur integrierter Informationssysteme
AWL	Anweisungsliste
Blob	Binary Large Object
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language (BPEL) for Web Services
BPMN	Business Process Modeling Notation
CAEX	Computer Aided Engineering Exchange
CPS	Cyber-Physical Systems
DEA	Deterministische Endliche Automaten
DPWS	Device Profiles for Web Services
EA	Endliche Automaten
EPK	Ereignisgesteuerte Prozessketten
ERP	Enterprise Resource Planning
ES	Engineering Station
FBS	Funktionsbaustein-Sprache
FDI	Field Device Integration
HMI	Human-Machine-Interface
IEC	International Electrotechnical Commission
ISA	International Society of Automation
ISO	International Organization for Standardization
K3	Koordination, Kooperation und Kommunikation
KOP	Kontaktplan
LIMS	Labor-Informations- und Managementsystem
MES	Manufacturing Execution System
MSR	Mess-, Steuerungs- und Regelungstechnik

NAMUR	Interessengemeinschaft Automatisierungstechnik der Prozessindustrie
NEA	Nichtdeterministische Endliche Automaten
OASIS	Organization for the Advancement of Structured Information Standards
OCEB	OMG Certified Expert in Business Process Management
OMG	Object Modeling Group
OPC UA	Open Platform Communications Unified Architecture
OS	Operator Station
OWL	Web Ontology Language
PCS	Process Control System
PFC	Procedural Function Charts
PID	Proportional-Integral-Differential
PIMS	Prozessdaten-Informations- und Managementsystem
PLC	Programmable Logic Controller
PLC SC	Programmable Logic Controller (PLC) Statecharts
plcML	PLC-Modeling Language
PLS	Prozessleitsystem
PLT	Prozessleittechnik
PN	Petrinetze
PNK	Prozessnahe Komponente
POE	Programm-Organisationseinheit
QoS	Quality of Service
R&I	Rohrleitungs- und Instumentrierungsdiagramm
RAMI	Referenzarchitekturmodell Industrie 4.0
RIO	Remote Input/Output
SC	Statecharts
SFC	Sequential Function Charts
SIPN	Steuerungstechnisch Interpretierbare Petrinetze
SOA	Service-oriented Architecture
SPS	Speicherprogrammierbare Steuerung
SSC	Sequential State Charts
ST	Strukturierter Text
SysML	Systems Modeling Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WS-BPEL	Web Service BPEL

WSDL	Web Service Description Language
XML	Extensible Markup Language
XSD	Extensible Markup Language (XML) Schema Definition
XSLT	Extensible Stylesheet Language Transformation

Kurzfassung

Prozeduren nehmen in der Automatisierung einen immer größeren Stellenwert ein. Prozeduren zum Start eines Motors werden ebenso wie Abläufe zur Steuerung von Geschäftsprozessen seit vielen Jahren betrachtet und in domänenspezifischen Sprachen beschrieben. Viele dieser domänenspezifischen Sprachen sind in der Praxis erprobt. Sie ähneln sich, besitzen jedoch keine gemeinsame Syntax und Semantik. Dies ist vor allem den unterschiedlichen Anforderungen der Domänen geschuldet. Während die Steuerungsprozedur einer Maschine formal definiert sein muss, damit die Steuerung sie eindeutig ausführen kann, liegt der Fokus einer Instandhaltungsprozedur auf einer Darstellung, die durch den menschlichen Akteur einfach verstanden und umgesetzt werden kann. Im Gegensatz zu Maschinen können Menschen auch informale Aufrufe interpretieren. Die bestehenden Prozedurbeschreibungssprachen werden in einer umfassenden Analyse betrachtet.

Die Vielfalt der domänenspezifischen Sprachen hat kein Problem dargestellt, solange die beschriebenen Prozeduren unabhängig voneinander betrachtet werden. Heutzutage wird allerdings unter den Schlagworten „horizontale Integration“ und „vertikale Integration“ eine ganzheitliche Betrachtung von Prozeduren angestrebt. Sowohl die horizontale als auch die vertikale Integration sind essentielle Bestandteile im Zukunftsprojekt Industrie 4.0. Zum einen sollen hierbei die Prozeduren zwischen den einzelnen Domänen ausgetauscht werden können, ohne dass jede Domäne die Sprachen der anderen Domänen verstehen muss. Zum anderen ist ein Zugriff auf die Informationen über den Prozedurzustand und die Beeinflussung von Prozeduren über die Ebenen der Automatisierungspyramide hinweg von Nöten.

In dieser Arbeit wird ein Referenzmodell zur Prozedurbeschreibung erarbeitet, das den gemeinsamen Kern der domänenspezifischen Sprachen beschreibt: Sie bestehen aus zwei Typen von Elementen, einer dieser Typen wirkt aktiv auf die Umgebung ein, der andere Typ reagiert auf Änderungen der Umgebung. Des Weiteren gibt es gemeinsame Konstrukte zur Modellierung von Hierarchien, Alternativverzweigungen und Nebenläufigkeiten. Die Interaktion mit der Umgebung ist einer der Hauptunterschiede zwischen den domänenspezifischen Sprachen. Damit auch in dieser Hinsicht ein universelles Modell erzeugt werden kann, muss die Interaktion mit der Umgebung über Dienstaufrufe und Zustandsabfragen erfolgen. Dies deckt das Setzen und Abfragen von Variablen, aber auch die informale Übermittlung eines Arbeitsauftrags des Chefs an seinen Mitarbeiter ab. Somit ist sichergestellt, dass das Prozedurmodell auf alle Komplexitätsgrade angewendet werden kann. Das Referenzmodell ist unabhängig von einer konkreten Visualisierung. Dies bietet den Vorteil, dass jeder Nutzer der Prozedurbeschreibung eine für ihn persönlich optimierte Darstellung auswählen kann, ohne dass eine Modifikation der Prozedurbeschreibung ist. Auch eine Kopplung mit Assistenzsystemen ist möglich.

Neben der Beschreibung der Prozedur wird ein Konzept benötigt, das die einheitliche und eindeutige Ausführung der Prozedur zulässt. Dies beinhaltet die operative Ausführung der Schrittkette im Regelfall, aber auch ein eindeutig definiertes Verhalten im Fehlerfall. Nur auf diese Weise ist eine Übertragbarkeit der Prozedur zwischen verschiedenen Systemen

gewährleistet.

Die Verwendung von Dienstaufrufen stellt ein Werkzeug zur Erhöhung der Flexibilität einer Prozedurbeschreibung dar. Die Abkehr von fest projektierten Signalverbindungen bewirkt, dass eine Zuordnung von benötigten Ausführungseinheiten erst zur Laufzeit erfolgen kann. Hier werden die kognitiven Fähigkeiten des Menschen ausgenutzt, die es ihm ermöglichen situationsbedingte Entscheidungen zu treffen. Ein Rollenkonzept unterstützt die flexible Zuordnung. Auf diese Weise kann der Entwickler der Prozedur in den Rollen Anforderungen an die Ausführungseinheiten definieren. Basierend auf den Rollen ist es realisierbar zur Laufzeit eine Zuordnung zu konkreten Ausführungseinheiten vorzunehmen.

Abstract

Procedures assume an increasingly important role within current automation technology. Procedures for starting an engine as well as sequences for controlling business processes have been examined and subsequently described in domain-specific languages for many years now. Many of these domain-specific languages are tried and tested in practice. Although the languages are fairly similar, they do not share a common syntax and semantics. This fact can be explained by the different requirements of the individual domains. While the control procedures of a machine must be formally described in order that the control system can execute them unequivocally. The focus of a maintenance procedure is firmly placed on a representation that can be easily understood and implemented by a human being. In contrast to machines, human beings can also interpret informal requests. The existing procedure description languages will be considered in a comprehensive analysis.

The variety of the domain-specific languages has never presented a problem as long as the described procedures were considered independently from each other. Nowadays, however, a holistic approach towards procedures is pursued under the headings of "horizontal integration" and "vertical integration". Both, the horizontal- and the vertical integration are essential components of the future-oriented project "Industrie 4.0". On the one hand, it is envisaged that procedures can be exchanged between individual domains without the necessity that each domain can understand the language of the other domain. On the other hand, it is required that information regarding a procedure's state can be accessed, and that procedures can be influenced by the different levels of the automation pyramid.

In this work, a reference model for procedure descriptions is developed that describes the common core of the different domain-specific languages. Basically, the languages consist of two types of elements. One of these element types actively influences the environment, while the other type reacts to changes in the environment. Additionally, there are shared constructs for the modelling of hierarchies, alternative branches and concurrencies. The interaction with the environment is one of the central differences between the domain-specific languages. In order to also create a universal model in this respect, the interaction with the environment can only be realized via service requests and status inquiries. This covers the setting and querying of variables as well as the informal submission of a task by the department head to his employees. This ensures that the procedure model can be applied to all levels of complexity. Moreover, the procedure model is independent of a concrete visualization. It is advantageous that each user of the procedure description can thus select a personally optimized representation without the need of having to modify the actual procedure description. A coupling with assistance systems is also possible.

Along with the description of the procedure, a concept that facilitates a consistent and unambiguous execution of the procedure is required. This includes the operative execution of the step chain during normal operations, but also a clearly-defined behavior in case of faults. This is the only way in which the portability of procedures between different systems can be ensured.

The use of service invocations can be understood as a tool to increase the flexibility of

the procedure descriptions. The renunciation of pre-configured signal connections has the effect that the allocation of the required execution units cannot occur until runtime. In this context, the cognitive abilities of human beings that enable us to make situation-based decisions can be fully exploited. The flexible allocation is additionally supported by a roles concept. By this means, the developer of a procedure can define the requirements of an execution unit in roles. Based on these roles, an allocation of the concrete execution units can be performed at runtime.

1. Einleitung

1.1. Motivation

Die systematische Steuerung von verfahrenstechnischen Produktionsprozessen tritt seit dem Beginn der Industrialisierung in den Vordergrund. In der vorindustriellen Gesellschaft wurde das Wissen über handwerkliche Prozesse innerhalb der Manufakturen im Vergleich zu heute ausschließlich mündlich weitergegeben. Seit der Erfindung der Dampfmaschine im 18. Jahrhundert wird das Gewerk der Mess-, Steuerungs- und Regelungstechnik (MSR) benötigt [140]. Für den Betrieb der Dampfmaschine wurden z. B. eine Schwimmerregelung für den Wasserstand und ein Zentrifugalregulator für den Dampfdruck entworfen [200]. Die MSR hat im Laufe ihrer Geschichte viele technologische Entwicklungen aus anderen Bereichen (z. B. Transistoren, Laser, integrierte Schaltkreise usw.) adaptiert. Aber auch gedankliche Konzepte, beispielsweise die Petrinetze oder die Objektorientierung, sind bereitwillig aufgegriffen worden. Der Einfluss der Informationstechnik begründete den Wandel von der klassischen MSR zur Prozessleittechnik (PLT) [140]. In Abbildung 1.1 sind typische Funktionen dargestellt, die neben der MSR zur PLT zählen.

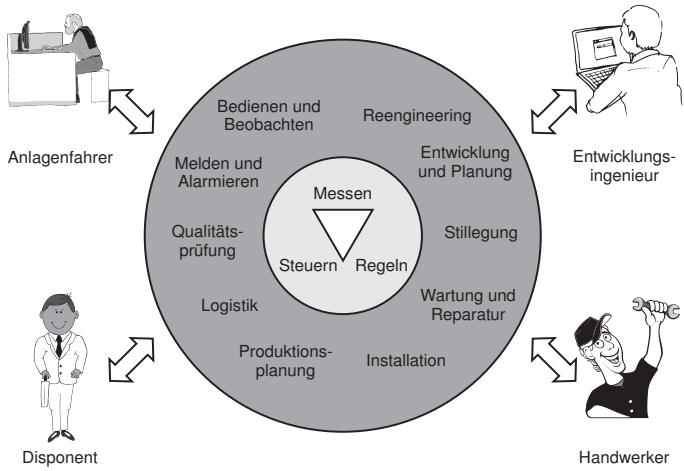

Abbildung 1.1.: Funktionen und menschliche Rollen in der Prozessleittechnik (angelehnt an [112, 149])

Viele der Funktionen, die in Abbildung 1.1 dargestellt sind, basieren auf Abläufen. Die

Bedeutung der Ablaufbeschreibung wird in Zukunft weiter zunehmen. So ist das Schaffen von

> *„Standards für eine die Automatisierungsebene übergreifende Prozedur- und Funktionsbeschreibung“ [91, S. 45].*

eine der Handlungsempfehlungen im Zukunftsprojekt Industrie 4.0. Beispielsweise wird in der Produktionsplanung die zeitliche Abfolge von Chargenprozessen festgelegt, Reparaturanleitungen erklären schrittweise die durchzuführenden Tätigkeiten oder in einer Steuerung sind Sequenzen zum Anfahren einer Anlage hinterlegt. Die Automatisierung solcher Steuerungsabläufe bietet verschiedene Vorteile, z. B. eine höhere Sicherheit und Zuverlässigkeit durch Vermeidung menschlicher Fehler, die Verbesserung von An- und Abfahrprozessen, eine bessere Wissenskonservierung und eine gesteigerte Effektivität der Anlagenfahrer [81]. Die Vollautomatisierung ist jedoch finanziell nicht zu realisieren, da Produktionsanlagen in der Prozessindustrie häufig einmalige und nur einmal errichtete Systeme sind [59]. Gerade in Ausnahme- und Fehlersituationen sind automatische Abläufe nicht implementiert, so dass der Mensch eingreifen muss. Durch eine Reduzierung des Implementierungsaufwands automatisierter Steuerungsabläufe wird es möglich sein, eine größere Anzahl zu automatisieren. Aufgrund immer komplexerer Anlagen und der Entfremdung des Anlagenfahrers vom Prozess (z. B. aufgrund von Remote Operation [9]) wird dies immer schwieriger [181]. Des Weiteren wird die Automatisierungslösung auf mehreren verteilten Automatisierungssystemen implementiert [44]. Der Ansatz der „Automatisierung der Automatisierung“ (vgl. z. B. [16, 146]) versucht möglichst viele menschliche Tätigkeiten im Umfeld der PLT zu automatisieren. Dennoch bleibt festzuhalten, dass auch in der Zukunft der Mensch eine entscheidende Rolle bei der Steuerung von Prozessen spielen wird [91] (vgl. Abbildung 1.1).

Abbildung 1.2 zeigt den Lebenszyklus einer Anlage.

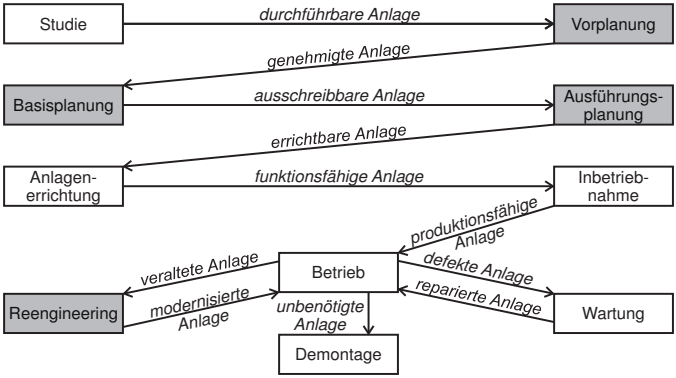

Abbildung 1.2.: Lebenszyklus einer verfahrenstechnischen Anlage (angelehnt an [99, 149])

Die Entwicklung von Automatisierungsfunktionen, d. h. die Erzeugung von Software im Steuerungssystem, findet hauptsächlich in den in Abbildung 1.2 grau hinterlegten Phasen der Vorplanung, der Basisplanung, der Ausführungsplanung und des Reengineerings

statt. Während der Entwicklung der Software werden aufgrund des Termindrucks häufig Adhoc-Lösungen implementiert. Diese führen häufig zu Unzufriedenheit bei den Betreibern, falls Änderungen bei der Inbetriebnahme oder während des Betriebs notwendig sind [161]. Adhoc-Lösungen lassen sich durch eine strukturierte Planung der Planungsprozesse weitestgehend vermeiden. Stattdessen soll ein durchgängiges Engineering etabliert werden, welches neben der vertikalen und der horizontalen Integration eines der wesentlichen Elemente von Industrie 4.0 ist [126]. Hinsichtlich der Steuerungsabläufe ist hier eine automatische Übernahme der Sequenzen aus der verfahrenstechnischen Planung zu nennen, die in der automatisierungstechnischen Planung weiter spezifiziert werden. Auch die Implementierung der Abläufe im Steuerungssystem soll aus der Planung ohne manuelle Tätigkeiten erstellt werden.

Eine Lösung hierfür ist das modellbasierte Engineering (vgl. z. B. [4, 51, 59, 186]). Das modellbasierte Engineering behebt Schwächen der konventionellen Softwareentwicklung. Dort ist z. B. die Qualität der Software maßgeblich vom Entwickler abhängig. Durch die nichtformalisierte textuelle Formulierung sind die Anforderungen nicht eineindeutig spezifiziert, die Anzahl der Fehler steigt mit der Komplexität und die Implementierung ist häufig die einzige Dokumentation [156]. Eine modellgetriebene Entwicklung bietet demnach eine Möglichkeit der Verkürzung und der besseren Verknüpfung der Entwicklungsphasen. Neben den Modellen zur Entwicklung der Automatisierungsfunktionen sind Modelle zur Kommunikation und zum Datenaustausch notwendig [16]. Andere Konzepte zur Verkürzung der Planungszeiten und damit eine schnellere Time-to-Market sind Package Units und modulare Anlagen. Hier besteht weiterhin ein hoher Forschungsbedarf, damit die nahtlose Integration in die Gesamtanlage funktioniert [133–135]. Die Unabhängigkeit von Beschreibungsaspekten ist z. B. eine Anforderung modularer Anlagen [134].

Neben dem Streben zu einem höheren Automatisierungsgrad ist eine flexible Reaktion auf sich ändernde Umgebungsbedingungen ein Ziel innerhalb der PLT. Unter Flexibilität wird hier die Möglichkeit verstanden Funktionen während des Lebenszyklus der Anlage zu ergänzen, zu modifizieren oder zu löschen. Diese Funktionen brauchen nicht bereits während der Planungsphase entwickelt zu werden [51]. Aus Anlagensicht kann dies durch die bereits erwähnten modularen Anlagen oder Mehrproduktanlagen realisiert werden. Aber auch flexiblere Steuerungssysteme sind notwendig. Hier besteht erhebliches Verbesserungspotential bei der Verknüpfung der leittechnischen Funktionen [71] (vgl. Abbildung 1.1). Beispielsweise verhindert eine statische Produktionsplanung schnelle Reaktionen auf Änderungen. Eine dynamische Produktion bedingt durch unvorhergesehene Bestelleingänge wird durch die Steuerungssysteme nicht ausreichend unterstützt [13]. Produkte müssen über ihren gesamten Lebenszyklus überwacht und nachverfolgt werden. Die Produktionsschritte müssen zunehmend nicht nur in qualifizierungspflichtigen Prozessen dokumentiert werden [87, 121]. In der Industrie 4.0-Initiative ist die Verflechtung von technischen Prozessen mit Geschäftsprozessen ein wesentliches Ziel [91, 154]. Schwankungen in den Preisen von Hilfs- und Rohstoffen machen kurzfristige Logistikvorgänge notwendig [74]. Die leittechnischen Funktionen müssen demnach interagieren, damit eine flexible Produktion ermöglicht wird. Dem stehen jedoch unterschiedliche Systeme [159], verschiedene Quellen zur Ermittlung des Bedarfs [161], mangelnder Zugriff auf Produkt- und Materialpreise [74] und ein fehlendes gemeinsames Begriffsverständnis [57] gegenüber.

Zusammenfassend bedeutet dies, dass Automatisierungsfunktionen zum einen einfacher geplant werden und zum anderen stärker mit anderen Funktionen vernetzt sein müssen. In [52] wird dies durch die zwei Herausforderungen „Virtualisierung der Infrastruktur“ und

„Funktionsadaption zur Laufzeit“ beschrieben. Hierzu reicht die Definition von Schnittstellen und Kommunikationskanälen nicht aus. Vielmehr ist ein gemeinsames Begriffsverständnis notwendig [57, 126], sowohl zwischen den verschiedenen Gewerken im Planungsprozess als auch zwischen den unterschiedlichen Sichten auf die Automatisierungsfunktionen. Auf diese Weise sind eine Verkürzung der Planungs- und Bauzeit durch einen Datenaustausch zwischen verschiedenen Gewerken, eine einfachere Datengewinnung für Wartungs- und Modernisierungsmaßnahmen und eine automatische Übernahme individueller Kundenwünsche möglich [126].

1.2. Zielsetzung

In dieser Arbeit werden drei Ziele verfolgt, eine Analyse bestehender Prozedurbeschreibungssprachen, die Erstellung eines Referenzmodells zur Prozedurbeschreibung und die beispielhafte Anwendung des Referenzmodells.

Umfassende Analyse bestehender Prozedurbeschreibungssprachen

Das erste Ziel dieser Arbeit besteht im Erstellen einer Übersicht über die bestehenden Prozedurbeschreibungssprachen. Diese Übersicht ist eine notwendige Basis für die Ermittlung des gemeinsamen Kerns der Beschreibungssprachen. Darauf aufbauend kann auch eine Entscheidung getroffen werden, welche Sprachen in einer konkreten Automatisierungsaufgabe verwendet werden können.

Definition eines Meta-Modells zur Prozedurbeschreibung

Hauptziel dieser Arbeit ist die Herleitung eines allgemeinen Referenzmodells zur Prozedurbeschreibung. Durch das Referenzmodell sollen zwei wesentliche Punkte adressiert werden, die in diesem Kapitel als verbesserungswürdig identifiziert wurden: Ein vereinfachter Entwurfsprozess von Steuerungsabläufen und das Schaffen eines gemeinsamen Verständnisses des Begriffs „Prozedur“ in den verschiedenen Gewerken im Umfeld der PLT. Mit dem Referenzmodell soll die gemeinsame Grundsemantik bestehender Prozedurbeschreibungssprachen abgebildet werden. Es muss eine Abstraktionsschicht bilden, die die kompakte Beschreibung der Prozeduren ermöglicht, ohne auf Sprachdokumentationen im Umfang mehrerer hundert Seiten zurückgreifen zu müssen. Das Referenzmodell soll explizit keine neue Prozedurbeschreibungssprache sein. Des Weiteren muss sich das Referenzmodell von der Modellierung von kontinuierlichen Regelungen und Funktionsbausteinen klar abgrenzen. Auch die Kommunikationstechnik zwischen Steuerung und Anlage sowie die Modellierung der Anlage sollen nicht behandelt werden.

Durch die Verwendung einer Service-oriented Architecture (SOA) zur Kommunikation zwischen Steuerung und Anlage und zwischen verschiedenen Steuerungssystemen soll eine Virtualisierung der Infrastruktur und eine Funktionsadaption zur Laufzeit ermöglicht werden. Durch die Idee der Dienstaufrufe werden (quasi-)kontinuierliche Steuerungsfunktionen von außen durch die Prozeduren steuerbar sein. Die Prozedur wird somit durch die Dienstaufrufe von der Realisierung der Funktionen getrennt werden. Des Weiteren soll die Interaktion zwischen menschlichen und maschinellen Akteuren im Produktionskontext erfasst, modelliert und gesteuert werden.

Untersuchung zur Anwendbarkeit des Meta-Modells

Das dritte Ziel besteht im Aufzeigen der Vorteile des entwickelten Referenzmodells. Dies soll an ausgewählten Beispielprozeduren geschehen, die einen Bogen von einer simplen Startprozedur eines Motors bis hin zu Prozeduren mit menschlicher Interaktion spannen. Auch der Entwurfsprozess einer Prozedur wird hierbei betrachtet werden.

1.3. Gliederung

Zu Beginn der Arbeit werden in Kapitel 2 grundlegende Begriffsdefinitionen vorgenommen. Hier wird ein besonderer Fokus auf das kybernetische Grundprinzip gelegt. Auf diese Weise wird eine saubere Trennung zwischen Prozess, Prozedur und Ausführungseinheit vorgenommen. Ebenfalls wird die Verteilung der Automatisierungsfunktionen auf die Ebenen der Automatisierungspyramide erläutert. Den Abschluss der Begriffswelt bildet die Einführung von Modellen als Prozedurbeschreibungsmittel.

In Kapitel 3 sind zunächst die grundlegenden Konzepte erläutert, die in Prozedurbeschreibungssprachen enthalten sind. Im Anschluss werden verschiedene Beschreibungssprachen für Prozeduren für technische Prozesse und Geschäftsprozesse betrachtet. Hierbei wird ein Mapping der Sprachen auf die Sprachelemente vorgenommen. Abschließend werden die Vorteile der Sprachen für ihre jeweilige Domäne erfasst und die Probleme im Zusammenwirken zusammengefasst.

Die in der Motivation genannten Herausforderungen werden in Kapitel 4 in Anforderungen an ein Referenzmodell zur Prozedurbeschreibung umgewandelt. Aufbauend auf den Anforderungen und der Analyse in Kapitel 3 wird dieses Referenzmodell anschließend als Meta-Modell entwickelt. Die Darstellungsformen durch ein meta-modellbasiertes Visualisierungssystem und als XML-Struktur runden die Modellvorstellung ab. Kapitel 5 enthält zwei Anwendungsbeispiele, die Steuerprozedur der Pumpe einer LKW-Abfüllung und eine Ventil-Wartungsprozedur.

Das erstellte Referenzmodell wird in Kapitel 6 zusammengefasst sowie anhand der Anforderungen und der umgesetzten Beispiele kritisch diskutiert. Ein Ausblick über weitere Forschungsaktivitäten bildet den Schluss der Arbeit.

2. Definitionen und grundlegende Begriffswelt

Die Begriffe Prozess, Prozessleitsystem, Kommunikation und Modellierung werden in vielen verschiedenen Literaturquellen verwendet und in all ihren Facetten beleuchtet. Dieses Kapitel dient nicht der detaillierten Darstellung der Literatur zu diesem Thema, sondern als Grundlage eines einheitlichen Begriffsverständnisses für den Rahmen der vorliegenden Arbeit.

2.1. Prozess

In Kapitel 1.1, S. 1, ist die Bedeutung von Prozessen für die industrielle Produktion dargestellt worden. Ein Prozess ist definiert als ein

> *„Satz von in Wechselbeziehung oder Wechselwirkung stehenden Tätigkeiten, der Eingaben in Ergebnisse umwandelt“ [82, S. 27].*

Die Prozessdefinition gilt allgemein für alle Arten von Prozessen. Der Prozessbegriff wird in verschiedenen Spezialisierungen verwendet (vgl. Abbildung 2.1).

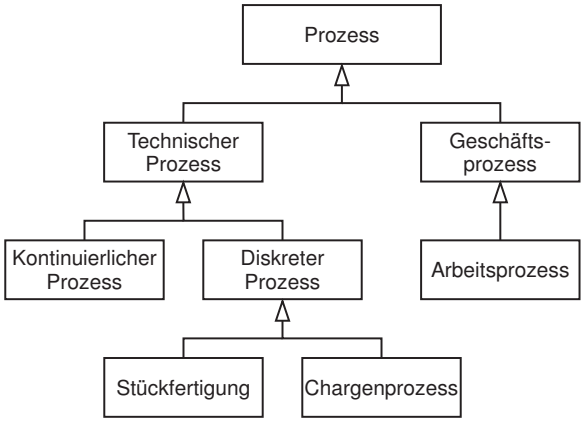

Abbildung 2.1.: Prozesstypen (nach [130])

Eine dieser Spezialisierungen ist der technische Prozess, welcher im Umfeld der Leittechnik von besonderem Interesse ist. Ein technischer Prozess umfasst die

> *„Gesamtheit der Vorgänge in einer technischen Anlage“ [34, S. 33],*

mit anderen Worten, ein technischer Prozess ist eine

> *„Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder Information umgeformt, transportiert oder gespeichert wird“ [54, S. 877].*

Technische Prozesse werden weiterhin in kontinuierliche und diskrete Prozesse unterteilt, wobei sich diskrete Prozesse in Stückfertigung und Chargenprozesse gliedern lassen. Für die Unterschiede zwischen den Spezialisierungen von technischen Prozessen sei auf [110, 137] verwiesen. Des Weiteren werden Prozesse ausgeführt, die den Zustand einer technischen Einrichtung ändern. Ein Beispiel hierzu ist ein Reinigungsprozess [81].

Neben den technischen Prozessen sind die Geschäftsprozesse eine weitere Spezialisierung von Prozessen. Ein Geschäftsprozess legt die Rahmenbedingungen für technische Prozesse fest und ist

> *„eine zielgerichtete, zeitlich-logische Abfolge von Aufgaben, die arbeitsteilig von mehreren Organisationen oder Organisationseinheiten unter Nutzung von Informations- und Kommunikationstechnologien ausgeführt werden können“ [68, S. 36].*

Arbeitsprozesse sind

> *„Geschäftsprozesse auf Mikroebene, die von Arbeitspersonen geplant, vollzogen, koordiniert und optimiert werden“ [145, S. 460].*

In der produzierenden Industrie werden alle Prozesse entweder mit dem Ziel der Wertschöpfung ausgeführt oder dienen zur Vorbereitung eines wertschöpfenden Prozesses. Anzumerken ist, dass sich Wertschöpfung nicht ausschließlich auf die Erzeugung eines verkaufsfähigen Produkts beschränkt. Auch Dokumente, Modelle und andere Artefakte zur Nutzung innerhalb eines Unternehmens stellen einen Wert dar [154]. Wertschöpfende Prozesse werden vom Menschen initiiert und müssen kontrolliert werden[1]. Die Kontrolle eines Prozesses wird im folgenden Unterkapitel behandelt.

2.2. Das kybernetische Grundprinzip

Wesentlich für das Verständnis dieser Arbeit ist das kybernetische Grundprinzip der Trennung von Steuerndem und Gesteuertem. Kybernetik ist die

> *„Wissenschaft von der Steuerung, d.h. der zielgerichteten Beeinflussung von Systemen“ [90].*

Ein System ist im Internationalen Elektronischen Wörterbuch definiert als

> *„Menge miteinander in Beziehung stehender Elemente, die in einem bestimmten Zusammenhang als Ganzes gesehen und als von ihrer Umgebung abgegrenzt betrachtet werden“ [34, S. 17].*

[1] Im Gegensatz hierzu stehen natürliche Prozesse. Natürliche Prozesse haben eine intrinsische Steuerung, die nicht beeinflusst werden kann und daher im Rahmen dieser Arbeit nicht behandelt wird.

Wird das kybernetische Prinzip auf die produzierende Industrie übertragen, läuft ein Produktionsprozess im Produktionssystem, das aus dem Produktsystem, dem Anlagensystem und dem Steuerungssystem besteht. Das Steuerungssystem wirkt auf das Anlagensystem und dieses wiederum auf das Produktsystem. Über die Rückwirkungen erhält das Steuerungssystem Informationen über den Prozess, die es verwerten kann (vgl. Abbildung 2.2).

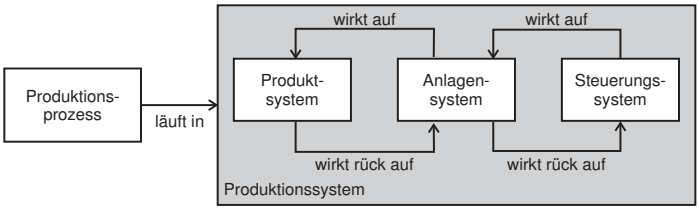

Abbildung 2.2.: Aufbau des Produktionssystems (nach [130])

Dieses Zusammenspiel zwischen Produktsystem, Anlagensystem und Steuerungssystem wird im Folgenden beschrieben.

2.2.1. Produktsystem

Ein Produkt ist

> *„etwas, was (aus bestimmten Stoffen hergestellt) das Ergebnis menschlicher Arbeit ist" [43].*

Ein Produkt kann als System betrachtet werden. Ein Produktsystem ist definiert als ein

> *„[m]aterielles oder immaterielles Objekt, das entsteht, um auf einem Markt zur Betrachtung oder zur Wahl, zum Kauf, zur Benutzung, zum Verbrauch oder zum Verzehr angeboten wird und geeignet ist, damit Wünsche oder Bedürfnisse zu befriedigen" [46, S. 381].*

Produktsysteme können demnach sowohl physische als auch gedankliche Gegenstände sein. Die Unterscheidung der verschiedenen Produkttypen ist nicht Bestandteil dieser Arbeit. Wichtig an dieser Stelle ist, dass ein Produkt bestimmte Eigenschaften besitzen muss, damit der wertschöpfende Prozess erfolgreich abgelaufen ist. Der Markt in der Definition aus [46] kann sich auch ausschließlich innerhalb des produzierenden Unternehmens befinden, wenn Produkte intern genutzt werden.

2.2.2. Anlagensystem

Ein technischer Prozess kann nach der Definition aus [34] nicht ohne eine zugehörige technische Anlage ausgeführt werden. Eine technische Anlage umfasst die

> *„Gesamtheit der technischen Einrichtungen und Vorrichtungen zur Bewältigung einer festgelegten technischen Aufgabe“ [34, S. 33].*

Zur Ausführung des Prozesses sind Objekte aus der physischen Welt notwendig, da erst eine technische Anlage dem Produktsystem eine definierte Umgebung bietet, in der sich das Produktsystem befinden kann [38].

Im Rahmen dieser Arbeit wird der Begriff Anlagensystem verallgemeinert und auf generelle Ausführungseinheiten erweitert. Eine Ausführungseinheit ist ein System, welches einen inneren Aufbau und einen Lebenszyklus besitzt. Eine Ausführungseinheit hat ein funktionales und kapazitives Leistungsvermögen. Sie steht mit anderen Objekten in einem Zusammenhang. Sowohl Menschen als auch Maschinen können Ausführungseinheiten sein [154].

Ausführungseinheiten können ein internes Steuerungssystem besitzen[2]. Sie können unterschiedliche Rollen einnehmen. Eine Rolle ist

> *„ein Element, das auf der einen Seite eine Realisierungseinheit in einem Modellsystem (Rollensystem) vertritt und auf der anderen Seite die Anforderungen an eine Realisierungseinheit spezifiziert“ [38, S. 40].*

Eine Rolle ist demnach im Kontext der Ausführungseinheiten eine Spezifikation von Anforderungen, die in einem bestimmten Produktionskontext durch die Ausführungseinheit erfüllt werden muss. Eine Rolle kann sowohl durch maschinelle als auch durch menschliche Ausführungseinheiten ausgefüllt werden [154].

Unterschiede zwischen maschinellen und menschlichen Ausführungseinheiten

Maschinelle Ausführungseinheiten können lediglich die bei ihrer Entwicklung vorgesehenen Aktionen ausführen [154]. Sie können rekonfiguriert werden, aber nicht flexibel auf neue Herausforderungen reagieren [169]. Menschliche Ausführungseinheiten hingegen haben zusätzlich die Möglichkeit auf unbekannte Situationen zu reagieren. Sie können also basierend auf ihrem Wissen und ihren Erfahrungen eigenständige Entscheidungen treffen [154]. Dies ist ein wichtiges Kriterium der Umsetzungsstrategie Industrie 4.0, die besagt, dass auch zukünftige hochautomatisierte Produktionsumgebungen immer Menschen bedürfen [125, 169]. Dort wird der Mensch als

> *„Dirigent im Wertschöpfungsnetzwerk“ [125, S. 48]*

bezeichnet. Menschliche Ausführungseinheiten sind von Natur aus sowohl flexibel als auch rekonfigurierbar [169].

Ausführungseinheiten müssen aufgerufen werden, damit sie Aktionen ausführen. Bei maschinellen Ausführungseinheiten muss der initiale Aufruf von extern kommen. Anschließend kann jedoch eine Ausführungseinheit weitere Aufrufe an andere Ausführungseinheiten schicken. Menschliche Ausführungseinheiten können sich im Gegensatz dazu selber initial aufrufen. Tritt beispielsweise ein Brand auf, startet ein Mensch selbstständig die Prozedur „Feuer löschen“ [154].

[2]Dies ist kein Widerspruch zu Abbildung 2.2, die Trennung zwischen Steuerndem und Gesteuertem ist bei internen Steuerungssystemen nur zu erkennen, wenn der interne Aufbau der Ausführungseinheit analysiert wird. Das System Ausführungseinheit wird dann in die Systeme interne Steuerungseinheit und Unter-Ausführungseinheit aufgeteilt.

Technische Agenten sind eine Mischung aus beiden Ausführungseinheitstypen. Unter einem technischen Agenten wird

> *„eine abgrenzbare (Hardware- oder/und Software-) Einheit mit definierten Zielen“ [174, S. 3]*

verstanden, die durch Reaktion auf ihre Umgebung in Zusammenarbeit mit anderen Agenten ein vorgegebenes Ziel erreicht. Agenten stellen demnach maschinelle Systeme dar, die flexibel Entscheidungen treffen können [174]. Auf technische Agenten wird im Rahmen dieser Arbeit nicht weiter eingegangen. Für zusätzliche Informationen hierzu sei auf [196] verwiesen.

2.2.3. Steuerungssystem

Analog zu den Ausführungseinheiten können sowohl Menschen als auch Maschinen Steuerungssysteme sein. Ein maschinelles Steuerungssystem ist ein

> *„Rechner- und Kommunikationssystem, in welchem ein Informationsprozess abläuft (Umformung, Verarbeitung und Transport von Information)“ [100, S. 6].*

Neben den Informationsprozessen, die in [100] genannt werden, ist auch die Speicherung von Informationen Aufgabe eines Steuerungssystems. Bei den klassischen maschinellen Steuerungssystemen wird zwischen Speicherprogrammierbarer Steuerung (SPS) und Prozessleitsystem (PLS) unterschieden.

Speicherprogrammierbare Steuerungen

Eine SPS, auf Englisch PLC, ist ein Computer in robuster Bauweise, der auf einem externen Programmiergerät in einer speziellen Programmiersprache nach der IEC 61131-3 [22] programmiert wird. In der Prozesstechnik werden sie meist in Kombination mit einem PLS verwendet. Typisch für eine SPS ist die zyklische Abarbeitung der Programme. In jedem Zyklus werden die Eingänge der SPS zunächst in das Prozessabbild eingelesen. Anschließend arbeiten die programmierten Steuerungsfunktionen auf dem Prozessabbild und schreiben die Ergebnisse ebenfalls in Variablen des Prozessabbilds. Im nächsten Schritt werden die Ausgänge der SPS entsprechend gesetzt [105].

Prozessleitsysteme

In der Prozesstechnik werden heutzutage üblicherweise verteilte PLS, auf Englisch Process Control System (PCS), verwendet (vgl. Abbildung 2.3).

Ein PLS interagiert über Feldgeräte mit dem zu steuernden Prozess. Ein Feldgerät kann entweder ein Sensor oder ein Aktor sein. Feldgeräte stellen die Schnittstelle zwischen Anlagensystem und Steuerungssystem dar. Feldgeräte sind mit einer Prozessnahen Komponente (PNK) verbunden. Die Verbindung ist entweder direkt, über eine Remote Input/Output (RIO) oder über weitere Sub-PNK realisiert [104]. Neben mindestens einer PNK besteht ein PLS aus mindestens einer Anzeige-/Bedienkomponente (ABK). Die ABK wird auch als Operator Station (OS) oder als Human-Machine-Interface (HMI) bezeichnet. Durch

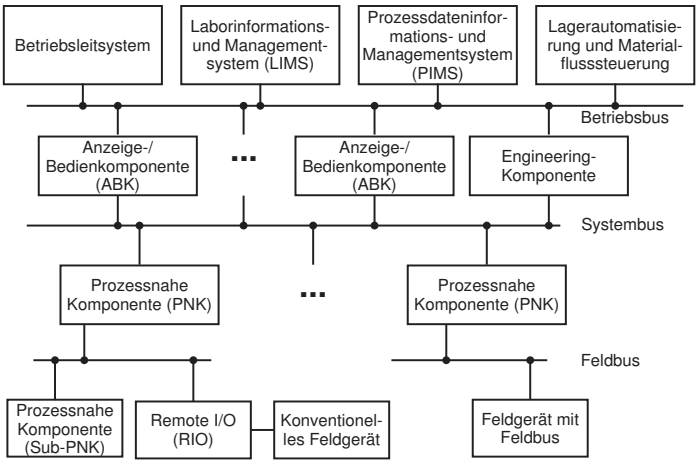

Abbildung 2.3.: Aufbau eines Leitsystems (nach [104, 164])

eine ABK wird der Anlagenfahrer über den Prozesszustand informiert und kann Handeingriffe in den Prozess vornehmen. Die ABK und die PNK sind über den Systembus miteinander verbunden, der häufig redundant ausgeführt wird und echtzeitfähig ist. Auf einer PNK werden die Steuerungsfunktionen ausgeführt. Es ist üblich, dass auch SPS als PNK verwendet werden. Konfiguriert werden die PNK über eine Engineering-Komponente (oder auch Engineering Station (ES) genannt). Neben den echtzeitfähigen Komponenten gibt es weitere, nicht-echtzeitfähige Systeme, die über den Betriebsbus angebunden sind. Hier sind z. B. das Betriebsleitsystem, das Labor-Informations- und Managementsystem (LIMS), das Prozessdaten-Informations- und Managementsystem (PIMS) und das Lager- und Materialfluss-Steuerungssystem zu nennen [164].

Der Mensch als Steuerungssystem

Nicht nur Automaten können als Steuerungssystem verwendet werden, auch Menschen können steuern. [154]. Beispielsweise empfängt ein Arbeiter Anweisungen von seinem Chef. In diesem Fall wirkt das Steuerungssystem „Chef“ auf die Ausführungseinheit „Arbeiter“ ein. Im direkten Produktionsumfeld greifen die Anlagenfahrer über die ABK in die Prozessteuerung ein [100]. Auch in zukünftigen Industrie 4.0-Umgebungen steht der Mensch als Steuerungssystem im Fokus [125].

2.3. Steuerungsfunktionen

Eine Steuerungsfunktion ist ein

> *„Vorgang in einem System, bei dem eine oder mehrere variable Größen als*

Eingangsgrößen andere variable Größen als Ausgangsgrößen aufgrund der dem System eigenen Gesetzmäßigkeiten beeinflussen“ [34, S. 125].

Steuerungsfunktionen werden in zwei Klassen aufgeteilt, zeitkontinuierliche und zeitdiskrete Steuerungsfunktionen [100]:

- Zeitkontinuierliche Steuerungsfunktionen führen Fließprozesse, deren Ein- und Ausgangsgrößen kontinuierlich sind. Dies können Regelungen mit einem geschlossenem Wirkungskreis (z. B. ein Proportional-Integral-Differential (PID)-Regler) oder Steuerungen mit einer offenen Wirkungskette (z. B. ein Kennfeld) sein. Kennzeichnend für zeitkontinuierliche Steuerfunktionsklassen ist die auf der Vorgabe eines Führungssignals basierende kontinuierliche Erzeugung eines Stellsignals. Bei einem geschlossenen Wirkungskreis geht zusätzlich der Anlagenzustand in die Berechnung des Stellsignals mit ein [100].
- Zeitdiskrete Steuerungsfunktionen reagieren auf diskrete Ereignisse. Dies können binäre Variablen (z. B. Behälter ist voll) [100], aber auch Vergleiche mit analogen Variablen (z. B. Durchfluss größer als 51/min) sein [106]. Hierbei wird zwischen Verknüpfungssteuerungen und Ablaufsteuerungen unterschieden. Verknüpfungssteuerungen verbinden die Eingangssignale mit booleschen Funktionen [100] oder Funktionsbausteinen [106] zur Berechnung der Ausgangssignale.

2.3.1. Die Prozedur als Steuerungsfunktion

In der vorliegenden Arbeit liegt der Fokus auf der zeitdiskreten Steuerung von Prozessen durch Ablaufsteuerungen. Die Ablaufsteuerung ist als

„Steuerung mit schrittweisem Ablauf, bei der der Übergang von einem Schritt auf den folgenden programmgemäß entsprechend den vorgegebenen Übergangsbedingungen erfolgt“ [34, S. 150]

definiert. Zwischen den Schritten einer Ablaufsteuerung befinden sich Transitionen, die die Weiterschaltung von einem Vorgänger- zu einem Nachfolgeschritt kontrollieren [100]. Solche Ablaufsteuerungen sind nicht nur für diskrete Prozesse oder Batchprozesse von Bedeutung, sondern auch für kontinuierliche Prozesse [88].

Die Abfolge von Schritten und Transitionen, die der Ablaufsteuerung zu Grunde liegt, wird als Prozedur bezeichnet. Der Begriff „Prozedur“ stammt vom lateinischen Wort „procedere“, auf Deutsch „vonstattengehen“, ab. Die Definition, wie sie im Duden zu finden ist, ist für die Beschreibung der Abfolge von Schritten und Transitionen nicht zielführend. Es ist unzureichend, eine Prozedur als

„Verfahren, (schwierige, unangenehme) Behandlungsweise“ [41, S. 814]

zu definieren, auch wenn implizit angedeutet wird, dass etwas mit einem Objekt (in diesem Fall mit einer Person) durchgeführt wird.

In der Informatik ist eine Prozedur eine

„in sich abgeschlossene Befehlsfolge mit meist eigenem lokalen Datenbereich, die an beliebigen Stellen eines übergeordneten Programms, des Haupt- oder Oberprogramms, wiederholt aufgerufen und ausgeführt werden kann“ [64, S. 74].

Ein Blick in die englische Literatur führt zu folgender Definition des Prozedurbegriffs:

> *„procedure [...] the action that you must take to do sth[.] in the usual or correct way“ [189, S. 490].*

Von dieser Definition ausgehend ist im technischen Umfeld eine Prozedur eine

> *„specification of a sequence of actions or activities with a defined beginning and end that is intended to accomplish a specific objective“ [81, S. 16].*

Eine Prozedur ist demnach eine Menge von auszuführenden Aktionen, die durch Bedingungen voneinander getrennt sind. Es gibt verschiedene Grundmuster von Prozeduren, z. B. lineare Ketten mit Anfang und Ende, zyklische Ketten oder Abläufe mit Verzweigungen. Des Weiteren ist festzuhalten, dass die Definition des Geschäftsprozess starke Ähnlichkeit zur Prozedurdefinition hat. Mit anderen Worten, der Begriff „Geschäftsprozess“ bezeichnet sowohl den Vorgang selber als auch die Steuerung des Vorgangs [154].

Je nach Steuerungssystem bzw. Anlagensystem (Mensch oder Maschine) werden unterschiedliche Anforderungen an die Flexibilität und den Spezifikationsgrad einer Prozedur gestellt (vgl. Abbildung 2.4).

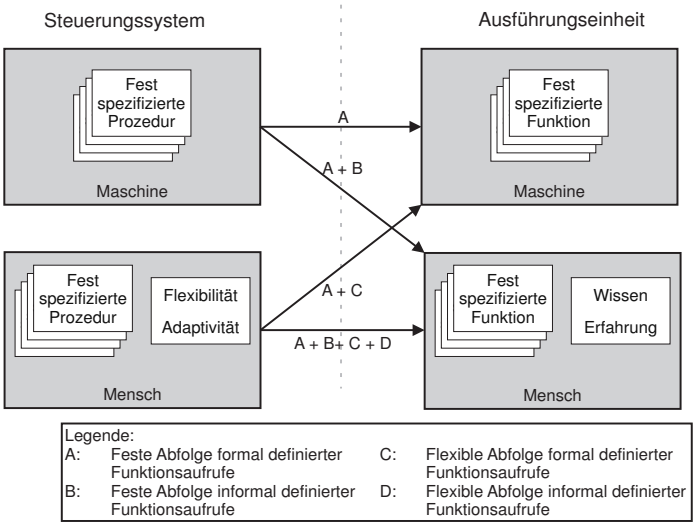

Abbildung 2.4.: Mensch und Maschine als Steuerungssystem und Ausführungseinheit

Wie bereits erwähnt besitzen maschinelle Ausführungseinheiten eine Menge fest spezifizierter Funktionen. Diese Funktionen sind rollenspezifisch und werden bei jedem Aufruf in genau der gleichen Weise ausgeführt. Menschliche Ausführungseinheiten haben ebenfalls solche fest spezifizierten Funktionen, können aber aufgrund ihres Wissens und ihrer Erfahrung jederzeit neue, unspezifizierte Funktionen entwickeln (rechte Seite in Abbildung 2.4).

In einem maschinellen Steuerungssystem sind fest spezifizierte Prozeduren gespeichert. Diese können auf Umgebungssituationen reagieren, alle Reaktionen müssen jedoch beim Design der Steuerung implementiert werden. Menschliche Steuerungssysteme können sich zusätzlich flexibel auf neue, unbekannte Situationen einstellen (linke Seite in Abbildung 2.4) [154].

Die Auswirkungen (Pfeile in Abbildung 2.4), die diese Eigenschaften der Ausführungseinheits- und Steuerungssystemtypen auf die Prozeduren haben, sind in Tabelle 2.1 erläutert.

Tabelle 2.1.: Interaktionsmöglichkeiten zwischen Steuerungssystem und Ausführungseinheit

		Ausführungseinheit	
		Maschine	Mensch
Steuerungssystem	Maschine	Ein maschinelles Steuerungssystem enthält in seiner fest spezifizierten Prozedur formale Funktionsaufrufe, die die maschinelle Ausführungseinheit verstehen und interpretieren kann.	Ein maschinelles Steuerungssystem enthält in seiner fest spezifizierten Prozedur formale und informale Funktionsaufrufe, die die menschliche Ausführungseinheit verstehen und interpretieren kann.
	Mensch	Ein menschliches Steuerungssystem kann in seinen fest spezifizierten oder flexiblen Prozeduren formale Funktionsaufrufe an die maschinelle Ausführungseinheit versenden, die dort interpretiert werden.	Ein menschliches Steuerungssystem kann in seinen fest spezifizierten oder flexiblen Prozeduren formale oder informale Funktionsaufrufe an die menschliche Ausführungseinheit versenden.

2.4. Funktionaler Leitsystemaufbau

Da es in einem Steuerungssystem verschiedene gleichzeitig ausgeführte Steuerungsfunktionen geben kann, müssen die Steuerungsfunktionen gegliedert und priorisiert werden. Ein solches gerichtetes Ordnungsschema wird als Hierarchie bezeichnet [38]. Bei einem PLS hat sich eine hierarchische Ebenenstruktur mit fünf Ebenen durchgesetzt (vgl. Abbildung 2.5) [27, 100, 163]. Dieser Aufbau wird als Automatisierungspyramide (AP) bezeichnet.

- Die Ebene 0 ist das Aktionsfeld, d. h., diese Ebene enthält den technischen Prozess, der auf der technischen Anlage ausgeführt wird [27, 100]. Sie gehört nicht direkt zum Leitsystem und wird daher in [163] auch nicht als Ebene gezählt.
- Ebene 1 ist die erste Ebene, die zum Leitsystem zählt. Sie wird auch als Feldebene bezeichnet und enthält alle Funktionen, die zum Bereich Messen, Stellen und Regeln gehören. Auch Verriegelungen und Schutzfunktionen gehören in diese Ebene [100].

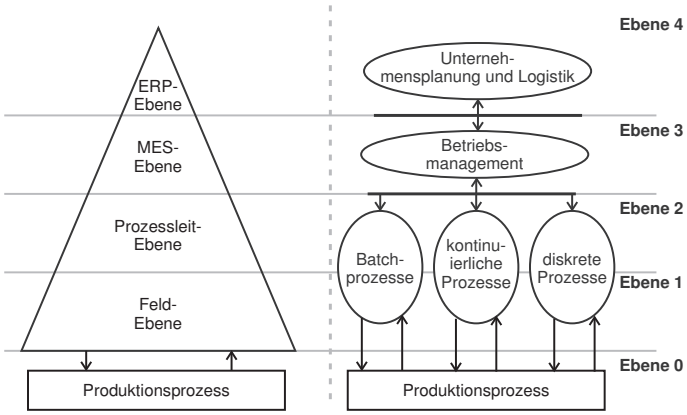

Abbildung 2.5.: Funktionale Hierarchie eines Leitsystems (nach [27, 163])

- In Ebene 2 ist die Prozessführung lokalisiert. Sie wird daher auch als Prozessleitebene bezeichnet. Neben der Prozessführung gehören auch die Prozessüberwachung, die Störungsbehandlung und die Prozeduren zum An- und Abfahren des Prozesses zur Ebene 2 [100]. Die Ebenen 1 und 2 sind echtzeitfähig, d. h., ihre Funktionen reagieren innerhalb einer vorgegebenen Zeit auf ein Ereignis. Des Weiteren werden viele Komponenten redundant ausgeführt, damit die Verfügbarkeit erhöht wird. In [27] werden die Ebenen 1 und 2 noch in die zu steuernden Prozesstypen Batchprozess, kontinuierlicher Prozess und diskreter Prozess aufgeteilt. Die Ebene 2 wird in [163] als Anlagenebene bezeichnet, beinhaltet aber die selben Funktionen.

- Ebene 3 umfasst die Betriebsablaufplanung sowie die Auswertung der Prozessergebnisse [27] und verknüpft die Geschäftsziele mit der Produktion [74]. Hier werden die produktionsrelevanten Geschäftsprozesse gesteuert [144]. Sie wird auch als Manufacturing Execution System (MES)-Ebene bezeichnet und in die Produktions- und Betriebsleitebene unterteilt. Typische Aufgaben sind die Datenerfassung und -analyse über die Produktion, über die Erstellung und Anpassung der lokalen Produktionsplanung, über Transport und Lagerung der Materialien oder die Kostenoptimierung eines Betriebskomplexes [27].

- Ebene 4, die Enterprise Resource Planning (ERP)-Ebene, enthält die Unternehmensführung. Hier werden Kostenanalysen und Auswertungen durchgeführt, welche die Unternehmensstrategie beeinflussen [100]. Dies betrifft z. B. die optimale Planung der Rohstoff- und Ersatzteilbestände, die Erfassung der Rohstoff- und Energieverbräuche, die Weitergabe von Personaleinsatzzeiten an die Personalabteilung oder die Erfassung von Qualitätsrückmeldungen von Kunden [27].

In klassischen PLS haben sich die Ebenen durch ihre Zeitskalen unterschieden. Während in Ebene 1 Sekunden, Millisekunden und manchmal sogar Mikrosekunden als Zeithorizont

benötigt worden ist, sind Eingriffe durch Ebene 2 im Minuten- bis Stundenbereich erfolgt. Aktionen der Ebene 3 haben sich in Tagen, Wochen und Monaten ausgewirkt. Strategieentscheidungen in Ebene 4 sind in den nächsten Monaten und Jahren relevant geworden [100]. Die heute üblichen direkten Zugriffe aus den höheren Ebenen auf die niedrigeren Ebenen der AP machen diese Zeitskalenunterscheidung jedoch obsolet [144].

Zwei der wesentlichen Vorteile des Ebenenmodells sind die Verdichtung von einer Vielzahl von Einzelinformationen von niedrigen zu hohen Ebenen sowie die Ableitung von Informationen für die niedrigen Ebenen durch die Vorgaben der höheren Ebenen [163]. Das Ebenenmodell hat sich daher in den letzten zwanzig Jahren bewährt. Dennoch wird unter dem Stichwort „vertikale Integration“ an der Struktur gerüttelt [144]. Sensoren, die bisher in der 4 . . . 20 mA-Technik[3] ausgeführt wurden, werden zunehmend von Sensoren mit digitaler Signalübertragung über einen Feldbus (vgl. Abbildung 2.3) verdrängt [71]. Sie besitzen interne Diagnosefunktionen und können über den Feldbus parametriert werden. Funktionen aus höheren Ebenen werden weiter unten implementiert [71] und sind daher nicht mehr nur einer Ebene zuzuordnen [144]. Des Weiteren führt eine permanent erforderliche Planungsbereitschaft dazu, dass MES-Systeme direkt auf Automatisierungssysteme der Feldebene zugreifen. Beide Punkte sind aufgrund der Weiterentwicklungen in der Informationstechnik möglich, erfordern aber eine Vielzahl von Schnittstellen. Zudem kann mit vielen Insellösungen, die für die einzelnen Anwendungsfälle implementiert werden, kein globales Optimum erzielt werden [144]. Allerdings hat diese Aufweichung der Automatisierungspyramide Grenzen [74]. Anforderungen bezüglich Echtzeit und funktionaler Sicherheit machen eine Verlagerung bestimmter Funktionen in höhere Ebenen schwierig bis unmöglich [182].

2.5. Kommunikation

Zur Informationsübertragung innerhalb eines PLS stehen eine Reihe von Methoden und Technologien zur Verfügung. Eine detaillierte Beschreibung der Kommunikationsmethoden würde den Rahmen der Arbeit sprengen, daher sei auf [48] für eine tiefere Erläuterung verwiesen. Aus diesem Grund wird hier nur die dienstbasierte Kommunikation als Möglichkeit der ebenenübergreifenden Integration [128] vorgestellt.

2.5.1. Dienstbasierte Kommunikation

Ein Dienst beinhaltet

> *„Funktionen, die einem Benutzer von einer Organisation angeboten werden“ [38, S. 50].*

Dienste können durch Menschen und durch Maschinen angeboten werden. Diese Definition ist generisch und kann sowohl auf technische als auch auf nicht-technische Systeme übertragen werden [38]. Abbildung 2.6 zeigt das Referenzmodell für ein Dienstsystem in der Prozessautomatisierung.

Die Kernidee des Diensts besteht in dem Abrufen von Funktionalität von einer anderen Stelle [18]. Dienste müssen nicht initialisiert werden, sie sind permanent verfügbar

[3] Die 4 . . . 20 mA-Technik übermittelt analoge Daten mittels eines Stromsignals, siehe [107].

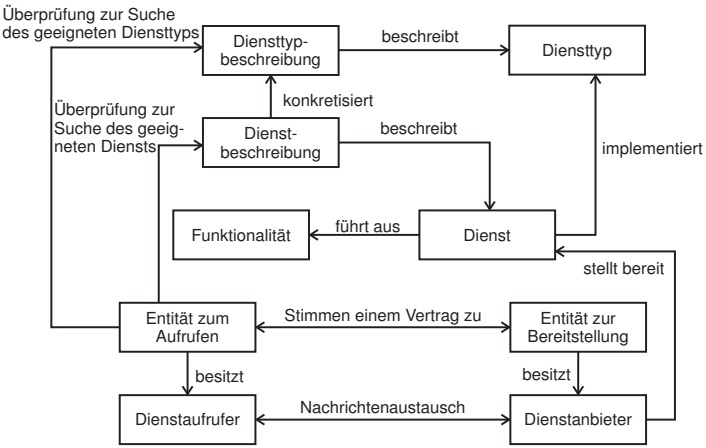

Abbildung 2.6.: Referenzmodell für ein Dienstsystem in der Prozessautomatisierung (nach [196])

[101]. Dabei ist das Wissen, wie eine Funktionalität realisiert wird, für den Dienstaufrufer nicht interessant [184]. Die Implementierung des Diensts verbleibt daher beim Dienstanbieter. Dieser stellt über eine Schnittstelle einen gekapselten Zugang bereit, über den der Dienstaufrufer die benötigten Daten an den Dienstanbieter senden kann. Ein Vertrag zwischen Dienstanbieter und -aufrufer beinhaltet die genaue Beschreibung der Dienstfunktionalität und der benötigten Daten sowie die garantierten nichtfunktionalen Eigenschaften [143]. Einen Überblick über verfügbare Dienste ist durch einen speziellen Verzeichnisdienst gegeben [101]. Ein Dienst realisiert einen Diensttypen. Im Diensttyp sind die Semantik, die Funktionen (z. B. Ein- und Ausgabe, Art der Funktionsaufrufe), das Umfeld des Dienstes und nichtfunktionale Qualitätsmerkmale enthalten [38].

Dienste ermöglichen den Wechsel von fest projektierten Kommunikationsbeziehungen kontextloser Daten zu einer losen Verbindung von Steuerungssystemen [18]. Anstelle von Signalen, die ausschließlich im Engineering der Steuerung festgelegt werden [198], erfolgt die Kommunikation hier durch diskrete Nachrichten [38]. Durch die Interpretation von Produktionsschritten als Dienste wird aus der klassischen prozedurorientierten Ablaufsteuerung die Orchestrierung einer Sequenz von Diensten [18]. Jeder Dienst besitzt einen eindeutigen Namen, der durch den Menschen interpretiert werden kann. Auf diese Weise ist eine eindeutige Identifikation des Diensts bei gleichzeitiger Lesbarkeit des Prozedurablaufs durch den Menschen gesichert [184].

Ein Dienst ist generell nicht an eine Kommunikationsinfrastruktur gebunden [38]. Dennoch ist die Festlegung auf eine bestimmte Infrastruktur notwendig, damit Nachrichten zwischen den beteiligten Dienstanbietern und -nutzern versendet werden können. Verschiedene Publikationen sehen Open Platform Communications Unified Architecture (OPC UA) als zukünftige Plattform, in der ein Dienstsystem in der industriellen Produktion imple-

mentiert werden kann (z. B. [17, 144]).

2.6. Prozedurbeschreibungsmittel

Ein Prozess ist ein einmaliger Vorgang, der in der physischen Welt zu einem definierten Zeitpunkt an einem definitiven Ort stattfindet [154]. Ein Prozess kann nicht rückgängig gemacht werden. Nachdem der Prozess beendet worden ist, sind nur die Auswirkungen des Prozesses noch zu beobachten. Die Auswirkung eines Prozesses kann durch einen weiteren Prozess rückgängig gemacht werden. Diese Eigenschaften des Prozesses können auch auf Prozeduren übertragen werden. Da nach Beendigung eines Prozesses nur die Auswirkungen sichtbar sind, müssen Prozesse überwacht werden. Die Ergebnisse der Überwachung werden in einer Prozessdokumentation festgehalten [38].

Für den Prozess, die Prozedur und die Dokumentation sind Beschreibungen notwendig (vgl. Abbildung 2.7). Anhand der Prozessbeschreibung werden eine Prozedurbeschreibung und eine Beschreibung der Prozessüberwachung erzeugt. Basierend auf der Prozedurbeschreibung wird die Prozedur im Steuerungssystem ausgeführt und der Prozess auf diese Weise gesteuert. Die Prozedurbeschreibung ist demnach ein zentrales Dokument, welches benötigt wird, damit Prozesse reproduzierbar und kontrolliert ausgeführt werden können.

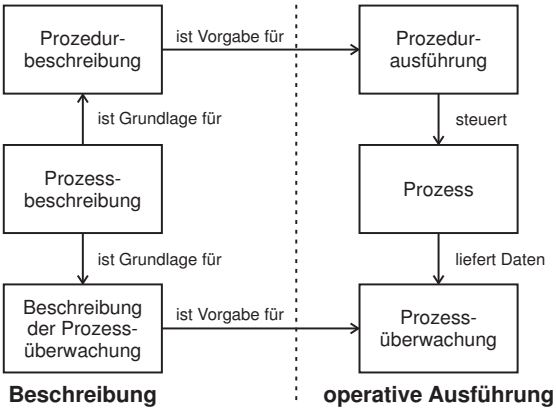

Abbildung 2.7.: Die Wesensart eines Prozesses (nach [38, 154])

Jede Form der Beschreibung benötigt ein Beschreibungsmittel, in dem die Beschreibung niedergelegt ist.

> *„Beschreibungsmittel definieren eine Menge von Zeichen, Symbolen und Regeln, die zur Modellierung eines Systems innerhalb eines bestimmten Kontexts zugelassen sind“ [115, S. 11].*

Beschreibungsmittel können formal, semiformal und informell sein [157]. Die Wahl des richtigen Beschreibungsmittels ist essentiell für die erfolgreiche Abstraktion des realen Systems

[115]. Neben formalen Sprachen, die aus Wörtern und Grammatik-Regeln bestehen, können Prozesse und Prozeduren durch Modelle beschrieben werden.

2.6.1. Formale Sprachen

Kommunikation erfolgt, damit Menschen Informationen zu anderen Menschen übertragen können. Menschen verwenden jeden Tag eine natürliche Sprache, damit sie mit anderen Menschen kommunizieren können. Ebenso verarbeiten Computer sprachliche Eingaben. Diese müssen in einer formalen Sprache formuliert sein. Natürliche Sprachen enthalten unvollständige, vage oder falsche Konstrukte, die zwar von Menschen verstanden werden können, nicht aber von Computern [15].

Eine Sprache ist als eine

> *„Menge von Wörtern über einem Alphabet Σ“ [10, S. 24]*

definiert. Ein Alphabet ist hierbei eine beliebige

> *„endliche, nichtleere Menge Σ [...]. Die Elemente eines Alphabets werden Buchstaben genannt“ [10, S. 15].*

Ein Wort über einem Alphabet Σ

> *„ist eine endliche (eventuell leere) Folge von Buchstaben aus Σ“ [10, S. 16].*

Wörter im Sinne dieser Definition sind nicht nur die Elemente, die umgangssprachlich als Wörter bezeichnet werden. Auch Sätze, Aussagen, Diagramme, Terme, Modelle usw. sind Wörter einer Sprache. Die Definitionen aus [10] gelten sowohl für natürliche als auch für formale Sprachen. Formale Sprachen sind

> *„künstlich entworfene Sprachen, die mit formalen Mitteln exakt zu beschreiben sind“ [79, S. 33].*

Formale Sprachen bestehen in der Regel aus abzählbar unendlich vielen Wörtern. Es wird daher eine endliche Menge an Bildungsregeln, die sogenannte Syntax, für die Definition formaler Sprachen benötigt [79]. Die Syntax einer Sprache teilt die Menge aller Wörter, die über einem Alphabet gebildet werden können, in zwei Teilmengen auf. Diese Teilmengen sind disjunkt, die eine Menge entspricht der Sprache, die andere dem Komplement der Sprache [192]. Erst durch eine Syntax werden aus Informationen Daten, die interpretiert und verstanden werden können [73].

Die Syntax alleine reicht nicht aus, damit eine formale Sprache als Kommunikationsmittel genutzt werden kann. Es ist ein Rückgriff auf die Semantik notwendig, die die Bedeutung der Wörter festlegt [79]. Mittels der Semantik werden syntaktisch korrekte Wörter einer Sprache auf eine semantische Domäne bezogen. Semantische Abbildungen werden häufig induktiv definiert [192]. Ferner muss die Formulierung einer Semantik mit großer Achtsamkeit erfolgen, da der Mensch dazu neigt, für ihn offensichtliche Punkte wegzulassen [73].

Neben textbasierten Sprachen existieren visuelle Sprachen. Eine visuelle Sprache besteht aus einer Menge von grafischen Elementen. Visuelle Sprachen nutzen die Eigenschaften des Menschen aus, der visuelle Informationen effektiver als textuelle Informationen aufnehmen und verarbeiten kann [192]. Des Weiteren vermuten die Nutzer von visuellen Sprachen,

dass diese einfacher als textuelle Sprachen zur Programmierung genutzt werden können und so ein produktiveres Arbeiten ermöglichen. Dabei gilt jedoch, dass visuelle Sprachen denselben Regeln genügen müssen wie textuelle Sprachen, damit sie von Computern verarbeitet werden können. Insbesondere hat eine gut gestaltete grafische Beschreibungssprache eine wohldefinierte und beschränkte Anzahl von Elementen, gute Abstraktionsmöglichkeiten und eine feste Basis in ihrer Syntax und Semantik und ist pragmatisch anwendbar [87].

Beschreibungssprachen werden insbesondere zur Modellierung verwendet. In diesem Zusammenhang werden sie als Modellierungssprache bezeichnet [192].

2.6.2. Modelle

Ein Modell ist ein

> *„Gegenstand, der es erlaubt Aussagen über einen anderen, modellierten Gegenstand zu treffen“ [38, S. 7].*

Modelle geben einen Ausschnitt aus der Realität wieder. Ein Modell ist nach [131, 162] durch die drei Eigenschaften Abbildungsmerkmal, Verkürzungsmerkmal und pragmatisches Merkmal charakterisiert:

- Die Eigenschaft Abbildungsmerkmal unterstreicht, dass ein Modell immer eine Abbildung eines natürlichen oder künstlichen Systems ist. Modelle können auch andere Modelle repräsentieren.
- Modelle geben im Allgemeinen nur eine Teilmenge der Eigenschaften des modellierten Systems wieder, sie verkürzen demnach.
- Modelle folgen einem pragmatischen Ansatz, sie sind für eine bestimmte Zeit, einen bestimmten Empfänger und für einen bestimmten Zweck vorgesehen.

Häufig werden Modelle mit mathematischen Modellen gleichgesetzt, die als formale Grundlage für Simulationen dienen [14]. In dieser Arbeit sind unter dem Begriff „Modell“ jedoch Modellsysteme gemeint. Ein Modellsystem ist ein

> *„Modell, das selbst als System strukturiert ist und das versucht den inneren Aufbau eines betrachteten Systems so gut nachzubilden, dass im gewünschten Kontext und mit der geforderten Genauigkeit die äußeren Eigenschaften des Modellsystems mit denen des betrachteten Systems übereinstimmen“ [49, S. 86].*

Ebenfalls von Bedeutung für die Modellierung sind die Festlegungen der Modellinteraktion [16]. Solche Modellsysteme können sowohl zur Analyse bestehender als auch zur Erstellung neuer Systeme genutzt werden [49]. Sie dienen der Verständnisbildung, dem Datenaustausch und der Datenhaltung [109].

Ein Modellsystem kann ein Referenzmodell sein. Ein Referenzmodell ist eine domänenspezifische Beschreibung. Die Beschreibung ist in sich schlüssig, allerdings kann es auch andere Varianten des Modells geben. Ebenso sind Gegenbeispiele möglich, auf die das Referenzmodell nicht angewendet werden kann. Ein Referenzmodell wird in der Fachwelt als beste der existierenden Modellvarianten angesehen. Kernmodelle sind spezielle Referenzmodelle. Sie sind universell gültige, domänenunabhängige Beschreibungen von Systemen.

Sie treffen Aussagen über das System, die „ewig“ gültig sind. Wird eine Aussage zu einem Kernmodell weggelassen, ist das Modell nicht mehr gültig [196].

Der Aufbau von Modellen wird durch Meta-Modelle definiert. Abbildung 2.8 zeigt die verschiedenen Ebenen der Modellierung. In einem Meta-Modell sind die Elemente, aus denen die abgeleiteten Modelle bestehen, sowie die Regeln zur Verwendung der Elemente enthalten. Die Beziehung zwischen dem Modell in der Typebene und dem modellierten Objekt in der Instanzebene muss nicht zwangsläufig im Meta-Modell enthalten sein. In Modellen sind nur die Elemente des Meta-Modells enthalten und sie müssen dessen Regeln genügen, damit das Modell gegenüber dem Meta-Modell valide ist [38]. Diese Aufteilung ist wiederholbar, so dass eine Meta-Meta-Modell-Ebene entsteht, die wiederum den Aufbau von Meta-Modellen beschreibt.

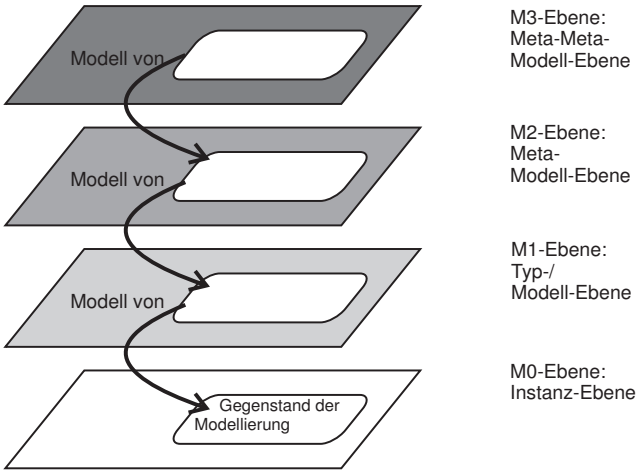

Abbildung 2.8.: Ebenen der Meta-Modellierung (nach [116])

Modelle werden im Umfeld der Automatisierungstechnik häufig verwendet. Bei der Programmierung lassen sich Fehler nicht vermeiden, die durch aufwändige Tests gefunden und anschließend behoben werden müssen. Daher werden Modelle erzeugt und verifiziert, die an verschiedene Situationen angepasst und folglich wiederverwendet werden können. Dieses Vorgehen ist unter dem Schlagwort *„Konfigurieren statt programmieren“* bekannt [144]. Es werden zwei verschiedene Vorgehensweisen unterschieden, modellgetriebene Codegenerierung und Modelle im Zielsystem:

- Bei der modellgetriebenen Codegenerierung werden die Modelle im Engineering-Prozess verwendet. Nach dem Engineering wird der Programmcode (z. B. für eine SPS) erzeugt und dieser kompiliert [183, 192].
- Im zweiten Fall sind die Modelle als erkundbare Struktur im Zielsystem vorhanden. Auch die Meta-Modelle sind in diesem Fall im Zielsystem vorhanden [51, 109, 185].

Die Verwendung von Modellen im Zielsystem hat mehrere Vorteile gegenüber der modellgetriebenen Codegenerierung. Zunächst ist die Anbindung von PNK an die höheren Ebenen der Automatisierungspyramide (siehe Abbildung 2.5, S. 15) einfacher. Die Anbindung kann durch abstrakte Modellverbindungen anstatt mit fest parametrierten Datenleitungen durchgeführt werden [109]. Auf diese Weise werden die Steuerungen flexibler und müssen daher seltener angepasst werden. Dies reduziert die Häufigkeit von Fehlern und die Anfälligkeit für schadhafte Manipulationen durch Angreifer. Ein weiterer Vorteil ist die Zuordnung gewonnener Daten zu Modellen. Hierdurch erhalten die Daten eine Semantik und können daher bei einem Funktionalitätswechsel einfacher an die neuen Steuerungsprogramme angepasst werden [185]. Der Nachteil besteht in der notwendigen Modifikation der PNK, damit sie mit Modellen arbeiten können. Aufgrund der langen Laufzeit von Prozessleitsystemen in der Prozessindustrie sind daher zwingend Überlegungen erforderlich, auf welche Weise die Modelle in existierende Systeme eingebracht werden können.

Modellierung mit Merkmalen

In vielen Fällen ist kein vollständiges Modell eines Systems erforderlich, sondern es sind lediglich bestimmte Eigenschaften des Systems relevant [50]. Aus diesem Grund werden den Systemen Merkmale zugeordnet:

> *„Merkmale sind ausgewählte Eigenschaften, die zur Klassen-, Begriffsbildung und Begriffsabgrenzung dienen. Merkmale sind eigene Objekte oder können durch Objekte repräsentiert werden.“ [112, S. 11].*

Merkmale sind demnach Eigenschaften, die ein System unter einem gewissen Betrachtungswinkel von anderen Systemen unterscheidbar machen. Sie ermöglichen eine semantisch lose Kopplung zwischen zwei Systemen. Ferner sind Merkmale nicht fest von Zuständen abgegrenzt. Je nach Betrachtungswinkel kann eine Eigenschaft als Zustand oder als Merkmal angesehen werden. Merkmale müssen jedoch im Gegensatz zu Zuständen während des Betrachtungszeitraums konstant bleiben, damit die Unterscheidung anhand des Merkmals möglich ist [50].

In [112] ist ein Merkmalmodell entwickelt worden, welches in Abbildung 2.9 dargestellt ist. Die grau hinterlegten Elemente sind Bestandteile auf der Meta-Modell-Ebene [92].

Ein realer Merkmalträger ist eine Entität, die außerhalb der Modellgrenze liegt. Diese kann durch mehrere Merkmalträger beschrieben werden. Diese Merkmalträger haben intern keine Beschreibung der inneren Funktionalität, diese wird durch den Merkmalträgertyp vorgegeben. Der Merkmalträgertyp enthält somit alle Merkmale, die zur Beschreibung eines Merkmalträgers benötigt werden. Der Merkmaltyp definiert sachneutral die Bedeutung des Merkmals [112].

Der Wert eines Merkmals wird auch Ausprägung des Merkmals genannt. Basierend auf den Merkmalen, die den Merkmalträgern zugeordnet wurden, können Aussagen über die Merkmale getroffen werden. Aussagen können entweder Merkmalträgern oder Merkmalträgertypen zugeordnet werden und durch ihren Aussagentyp klassifiziert werden. So können gegenständliche Aussagen (z. B. Messung, Simulation, Berechnung, Schätzung) oder nicht-gegenständliche Aussagen (z. B. Zusicherung, Anforderung, Festlegung oder Annahme) getroffen werden [112].

Das Merkmalmodell ermöglicht zunächst die Erstellung von Rollen, die abstrakt Anforderungen und Zusicherungen unabhängig vom konkreten Objekt festlegen. Somit liegt eine

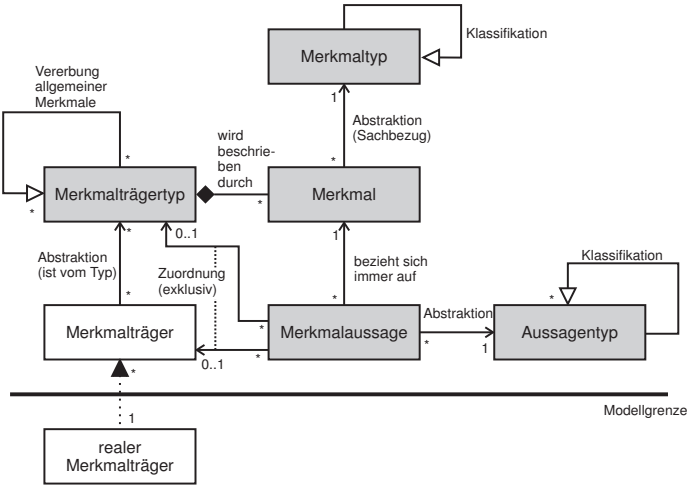

Abbildung 2.9.: Merkmalmodell (nach [92, 112])

Beschreibung unabhängig vom konkreten zu modellierenden System, dem Merkmalträger, vor. Erst bei der Zuweisung des Merkmalträgers wird der Wert des Merkmals zugeordnet [112].

3. Analyse von Prozedurbeschreibungssprachen

Die verschiedenen Typen von Steuerungssystemen und Ausführungseinheiten (vgl. Kapitel 2.3.1, S. 12) haben zu einer Vielzahl von Sprachen zur Prozedurbeschreibung geführt. Zudem sind durch die Weiterentwicklung der Technik fortlaufend neue Konzepte in die Prozedurbeschreibungssprachen eingeflossen. Diese Prozedurbeschreibungssprachen sind bereits an verschiedensten Stellen untersucht, verglichen und bewertet worden. Im folgenden Kapitel steht die Analyse hinsichtlich der Elemente einer Prozedurbeschreibungssprache nach Kapitel 3.1, S. 24, im Vordergrund. Die hier vorgestellten Ergebnisse basieren auf eigenen Analysen, die zu Teilen bereits in „Ein Referenzmodell zur Prozedurbeschreibung - Eine Basis für Industrie 4.0“ [152] veröffentlicht worden sind.

Die vorgestellten Prozedurbeschreibungssprachen sind grafische[1] domänenspezifische Sprachen (vgl. [84]). Durch sie wird das Verhalten einer Steuerung spezifiziert. Teilweise werden sie ebenfalls zur Implementierung verwendet. Die Spezifikation der Prozedur ist abstrahiert von der Implementierung bzw. der technischen Realisierung [6]. Die Prozedurbeschreibungssprachen sind kontrollfluss- und objektorientierte Modellierungssprachen [131], die den Wechsel von aktiven Zuständen beim Eintreten bestimmter Ereignisse beschreiben [113].

Grafische Prozedurbeschreibungssprachen sind Graphen. Ein Graph ist definiert als

> *„ein Tupel $G = (V, E, \Sigma)$ von einer Menge V von Knoten, einer Menge Σ von Gewichten mit $V \cap \Sigma = \emptyset$, [und] einer Menge $E \subseteq V \times \Sigma \times V$ von gewichteten Kanten“ [141, S. 29].*

Bipartite Graphen sind Spezialfälle von allgemeinen Graphen, deren Knoten sich so in zwei Gruppen einteilen lassen, dass jede Kante immer von einem Element aus einer Gruppe zu einem Element aus der jeweils anderen Gruppe führt [8].

Neben der Eigenschaft, dass Prozedurbeschreibungen Graphen sind, gibt es grundlegende Konzepte, die sich trotz des unterschiedlichen Anwendungszwecks in allen Sprachen finden lassen. Diese Konzepte werden im folgenden Abschnitt vorgestellt.

3.1. Elemente einer Prozedurbeschreibungssprache

In einer Prozedurbeschreibungssprache müssen Aussagen zu verschiedenen Konzepten getroffen werden. In Abbildung 3.1 sind generische Konzepte dargestellt, die eine Prozedur charakterisieren, nämlich: das Aufbaumodell, das Hierarchie- und Vernetzungsmodell, das Abstraktions- und Zuordnungsmodell, das Aktions- und Aktivitätenmodell sowie das

[1] Ausgenommen ist nur die Business Process Execution Language, die keine grafische Repräsentation besitzt. Sequential Function Charts besitzen neben der grafischen ebenfalls eine textuelle Repräsentation.

Ausführungssteuerungsmodell. Diese Konzepte sind in jeder Prozedurbeschreibungssprache vorhanden, lassen sich jedoch nicht immer explizit wiederfinden. Gerade während der Ausführung einer Prozedur sind sie oft verborgen.

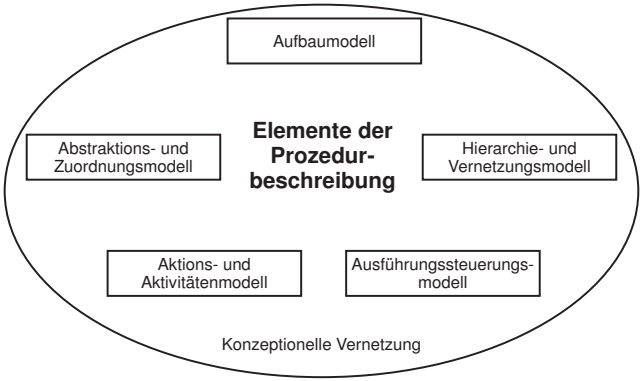

Abbildung 3.1.: Elemente einer Prozedurbeschreibungssprache (nach [130])

3.1.1. Aufbaumodell

Das Aufbaumodell beschreibt die grundlegenden Elemente, die eine Prozedurbeschreibungssprache beinhaltet. Beispiele für diese Elemente sind Schritte, Transitionen oder Zustände. Sie werden im Aufbaumodell, welches die Grundlage für alle weiteren Modelle bildet, in einen begrifflichen Zusammenhang gesetzt. Die Elemente bilden einen wesentlichen Teil der Knotenmenge der Graphdarstellung einer Prozedur. Nicht Bestandteil des Aufbaumodells sind die Verbindungsmöglichkeiten zwischen den Elementen oder die Art und Weise der Interaktion mit der Umgebung.

3.1.2. Hierarchie- und Vernetzungsmodell

Wie bereits erklärt, werden die Elemente des Aufbaumodells durch Kanten zu einem Graphen verbunden. Neben dem einfachsten Grundmuster einer Ablaufstruktur, einer linearen, terminierenden Kette, gibt es auch komplexere Ablaufstrukturen. Beispiele hierfür sind Verzweigungen, Makroschritte und Unterketten. Muster und Regeln zur Gestaltung all dieser vernetzten Ablaufstrukturen werden im Hierarchie- und Vernetzungsmodell beschrieben. Insbesondere beschreibt das Konzept, welche Alternative bei mehreren erfüllten Transitionsbedingungen einer Alternativverzweigung ausgewählt wird oder wie nebenläufige Ablaufketten beschrieben werden. Das Hierarchie- und Vernetzungsmodell beschreibt zudem den Aufbau verschiedener Ebenen. Dies beinhaltet zum einen den Aufruf von Unterprozeduren, zum anderen Makroschritte. Das Ausführungssteuerungsmodell beinhaltet

Aussagen über die Interaktion zwischen aufrufender und aufgerufener Prozedur, diese sind nicht Bestandteil des Hierarchie- und Vernetzungsmodells.

3.1.3. Abstraktions- und Zuordnungsmodell

Das Abstraktions- und Zuordnungsmodell beschreibt den Entwurfsprozess einer Prozedurbeschreibung. Zunächst betrifft dies den Spezifikationsgrad der Prozedurbeschreibung. In vielen Fällen wird die Prozedur bis hin zu einem vollständig spezifizierten Setzen von Variablen während der Ausführung der Prozedur spezifiziert. In anderen Fällen legt die Prozedurbeschreibung lediglich die Anforderungen fest, welche die Ausführungseinheit erfüllen muss. Mit anderen Worten, es muss determiniert sein, wie die Prozedurbeschreibung mit Rollen (vgl. Kapitel 2.2.2, S. 8) umgeht. Da Zuordnungen zwischen Rolle und Ausführungseinheit teilweise erst während der Prozedurausführung getroffen werden, müssen im Abstraktions- und Zuordnungmodell strukturell konsistente Transformationen beschrieben sein.

Ebenso ist ein Konzept zur Entwicklung von Prozedurklassen im Abstraktions- und Zuordnungsmodell enthalten. Bei Funktionsbausteinen hat sich ein Typ-Instanzenkonzept durchgesetzt (vgl. [47]). Die Funktionsbausteintypen werden vom Steuerungssystemhersteller implementiert. Während des Entwurfsprozess können Instanzen der Klassen verschaltet werden. Prozedurbeschreibungen hingegen sind nur in bestimmten Fällen wiederverwendbar. Für viele Anwender ist ein Typ-Instanzenkonzept vom Arbeitsaufwand her in der Regel nicht attraktiv. Sie verwenden stattdessen häufig Entwurfsmuster. Allerdings bringt die Verwendung eines Typ-Instanzenkonzepts Vorteile, beispielsweise bei der Erstellung von Batch-Rezepten oder der Übertragung einer Prozedur auf mehrere parallele Anlagenstraßen. Aus diesem Grund ist es sinnvoll, die Verwendung durch ein einfach anwendbares Konzept zu unterstützen.

3.1.4. Aktions- und Aktivitätenmodell

Eine Prozedur kann über Aktionen und Aktivitäten auf die Umgebung einwirken. Damit Informationen vom Steuerungssystem zum Empfänger gelangen, sind Kommunikationsbeziehungen erforderlich. Im Aktions- und Aktivitätenmodell sind diese Interaktionen der Prozedur mit der Umgebung festgelegt. Eine Aktion ist ein technologisch zeitloser Vorgang, während eine Aktivität eine Dauer besitzt [47]. Ein Beispiel für eine Aktion ist das Setzen einer Variablen, die Berechnung einer mathematischen Formel ist hingegen eine Aktivität. Der Aufruf eines Dienstes ist eine Aktion, falls das Steuerungssystem nicht auf eine Antwort wartet, anderenfalls ist der Aufruf eine Aktivität.

Die Interaktion kann beispielsweise verbal, durch Signale oder durch Dienstaufrufe ausgeführt werden. Ausnahmslos muss eine formale Beschreibung der Aktionsaufrufe vor dem Start einer Interaktion definiert werden, damit sich die Steuerungssysteme untereinander ebenso wie Steuerungssysteme und Ausführungseinheiten verstehen können. Bei menschlichen Kommunikationspartnern muss zumindest die eindeutige Interpretierbarkeit der Aktionsaufrufe sichergestellt sein.

3.1.5. Ausführungssteuerungsmodell

Das Ausführungssteuerungsmodell enthält die Beschreibung der operativen Ausführung einer Prozedur. Dies umfasst die Abarbeitung der Prozedur im Regelfall. Konkret sind hier dementsprechend Regeln festgehalten, welche vorgeben, in welcher Reihenfolge die entsprechenden Elemente des Aufbaumodells bearbeitet und wie die Aktionen und Aktivitäten des Aktions- und Aktivitätenmodells ausgeführt werden. Ebenso enthält das Ausführungssteuerungsmodell die Eingriffsmöglichkeiten durch externe maschinelle und menschliche Steuerungssysteme. Hier sind auch standardisierte Muster zur Ausnahmebehandlung als Zustandsmaschine hinterlegt, d. h., es liegt eine Beschreibung der Reaktionen der Prozedur auf Abweichungen vom Regelfall vor.

3.2. Auswahl der Prozedurbeschreibungssprachen

Insgesamt sind vierzehn Sprachen für die Analyse ausgewählt worden. Die Auswahl basiert auf [103, 151]. Viele der hier vorgestellten Sprachen sind durch die International Electrotechnical Commission (IEC), durch die International Organization for Standardization (ISO) oder durch die Object Modeling Group (OMG) standardisiert. Bei den Sprachen der OMG wird die Systems Modeling Language (SysML) nicht explizit betrachtet, da die Erweiterungen der SysML fast ausschließlich strukturelle Diagramme betreffen, welche statische Beziehungen modellieren. Eine Ausnahme liegt beim Aktivitätsdiagramm vor, welches in Kapitel 3.4.1, S. 54, erläutert wird [4]. Eine Betrachtung von Anwendungsdiagrammen und Sequenzdiagrammen im Rahmen der Verhaltensdiagramme der Unified Modeling Language (UML) ist nicht erforderlich, da dort der prozedurale Ablauf nicht im Vordergrund der Modellierung steht [11]. In [81] wird ein Konzept zum Entwurf von Ablaufsteuerungen von Konti-Anlagen aufgezeigt, jedoch keine eigene Beschreibungssprache definiert. Die formalisierte Prozessbeschreibung [176] kann als Grundlage für den prozeduralen Steuerungsentwurf dienen. In [172] ist gezeigt, dass aus formalisierten Prozessbeschreibungen automatisch Sequential Function Charts (SFC) erzeugt werden können. Da die formalisierte Prozessbeschreibung vorwiegend zur Beschreibung von Prozessen genutzt wird, wird sie nicht als Prozedurbeschreibungssprache betrachtet.

Jede einzelne der ausgewählten Sprachen wird zunächst hinsichtlich der allgemeinen Idee und des Einsatzzwecks vorgestellt und im Anschluss erfolgt die Ermittlung der fünf Elemente einer Prozedurbeschreibungssprache nach Kapitel 3.1, S. 24, in der jeweiligen Beschreibungssprache. Anzumerken ist, dass sich die Prozedurbeschreibungssprachen in zwei Gruppen einteilen lassen [152]: Zunächst werden Prozedurbeschreibungssprachen zur Steuerung von technischen Prozessen vorgestellt, anschließend solche, die Geschäftsprozesse steuern.

3.3. Prozedurbeschreibungssprachen zur Steuerung von technischen Prozessen

Der Fokus bei Prozedurbeschreibungssprachen zur Steuerung von technischen Prozessen liegt in der Interaktion von maschinellen Steuerungssystemen mit maschinellen Anlagensystemen. Es folgt die Vorstellung von neun Sprachen, Endliche Automaten (EA), Grafcet,

Grafchart, Procedural Function Charts (PFC), Petrinetze (PN), PLC Statecharts, SFC, Sequential State Charts (SSC) und Zustandsdiagramme, auf Englisch Statecharts (SC). Diese Sprachen sind aufeinander basierend entwickelt worden. Abbildung 3.2 gibt einen Überblick über die Abhängigkeiten der einzelnen Sprachen.

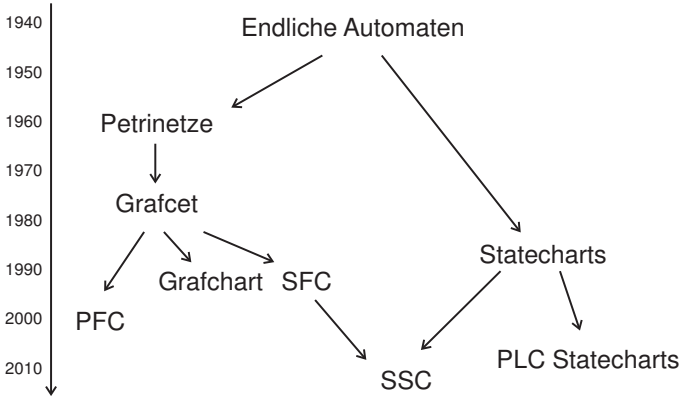

Abbildung 3.2.: Historie der Prozedurbeschreibungssprachen für technische Prozesse (basierend auf [196])

Zwischen SC-basierten Sprachen und PN-basierten Sprachen besteht ein grundlegender Unterschied in der Modellierung. Bei SC-basierten Sprachen ist der aktuelle Zustand der Prozedur durch das einzige aktive Element festgelegt. Bei PN-basierten Sprachen ist hingegen die Verteilung der Marken auf die Stellen für die Definition des Zustands der Prozedur relevant [6].

Im Folgenden werden die einzelnen Sprachen analysiert. Dabei werden ausgehend von den EA zunächst die PN-basierten Sprachen eingeführt, gefolgt von den SC-basierten Sprachen. Zum Schluss erfolgt die Betrachtung von SSC als Kombination beider Sprachfamilien.

3.3.1. Endliche Automaten

Endliche Automaten (EA) sind ein Beschreibungsmittel der Informatik aus den 1940er und 1950er Jahren [78]. Etymologisch stammt die Bezeichnung „Automat“ vom griechischen Wort $\alpha\upsilon\tau o\mu\alpha\tau o\sigma$ (automatos, übersetzt: von selbst geschehend) ab. In Abbildung 3.3 ist ein Beispiel für einen EA abgebildet, welches in den folgenden Absätzen erläutert wird.

Es existieren viele verschiedene Ausprägungen der EA. Der Hauptunterschied der Ausprägungen liegt in der Anbindung an die Umgebung [175]. EA sind ein geeignetes Mittel zur transparenten Beschreibung von Abläufen in der Automatisierungstechnik [62]. Sie bilden daher direkt oder indirekt die Grundlage für alle weiteren Prozedurbeschreibungssprachen in diesem Abschnitt.

Aufbaumodell Zunächst werden EA in Deterministische Endliche Automaten (DEA) und Nichtdeterministische Endliche Automaten (NEA) unterteilt. Ein DEA ist definiert als ein

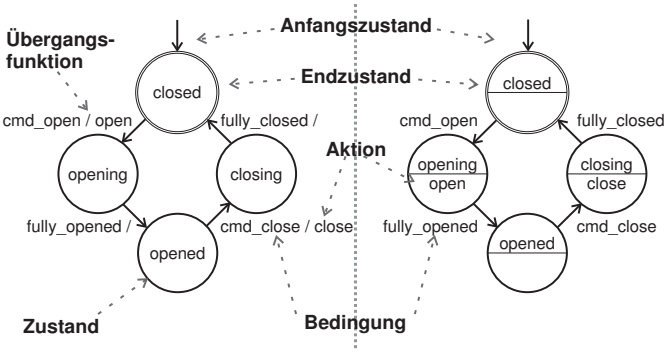

Abbildung 3.3.: Elemente eines endlichen Automaten (links ein Mealy-Automat, rechts ein Moore-Automat, nach [196])

Tupel $(Q, \Sigma, \delta, q_0, F)$. Hier ist

- $Q = \left\{q_1, \ldots, q_{|Q|}\right\}$ eine endliche[2] Menge von Zuständen,
- Σ eine Menge von Eingangssymbolen (ein Alphabet nach Kapitel 2.6.1, S. 19),
- $\delta : Q \to Q$ die Übergangsfunktion zwischen den Zuständen,
- $q_0 \in Q$ der Startzustand und
- $F \subseteq Q$ die Menge der Endzustände [78].

Ein Zustand ist als Startzustand ausgezeichnet. Es kann beliebig viele Endzustände geben [175]. Der Unterschied zwischen DEA und NEA besteht in der Übergangsfunktion. Bei einem DEA gibt δ einen einzelnen Zustand, bei einem NEA eine Menge von Zuständen zurück. Allerdings lässt sich zeigen, dass sich jeder NEA durch einen äquivalenten DEA ausdrücken lässt, so dass im Folgenden nur noch der DEA betrachtet wird [78].

Hierarchie- und Vernetzungsmodell Neben einer endlichen Menge diskreter Zustände besteht ein DEA aus möglichen Übergängen zwischen den Zuständen. Die Übergänge sind durch die Übergangsfunktion $\delta : Q \to Q$ definiert. Auch ein Übergang zum aktuellen Zustand zurück ist ein Übergang, der in δ enthalten sein muss.

Klassische EA unterstützen keine Hierarchien [175], d. h., alle Zustände müssen mit ihren Verbindungen dargestellt werden. Somit ist weder eine Zusammenfassung noch eine Verfeinerung von Zuständen möglich. Nebenläufigkeit wird in den EA ebenfalls nicht unterstützt[3].

[2] Daher die Bezeichnung „*Endlicher* Automat“.

[3] Einen Ansatz zur Erweiterung der EA zeigt [42], die detaillierte Darstellung würde jedoch den Rahmen dieser Zusammenstellung sprengen.

In [62] ist die Möglichkeit der Synchronisation mehrerer Automaten zur Modellierung von Nebenläufigkeiten erläutert. Das Fehlen von Hierarchien und Nebenläufigkeiten im Beschreibungsmittel führt jedoch zu einer Zustandsexplosion [72]. Zwei EA sind orthogonal zueinander, wenn sie unterschiedliche Aspekte eines Systems beschreiben, die keine Schnittmenge haben. Da EA keine Hierarchien kennen, werden die Zustände kombiniert. Seien z. B. E_1 und E_2 zwei orthogonale EA mit den Zuständen $q_{1,1}, q_{1,2}, \ldots, q_{1,m}$ bzw. $q_{2,1}, q_{2,2}, \ldots, q_{2,n}$. Die Summe der Zustände der beiden Automaten ist demnach $m + n$. Der kombinierte Automat besitzt jedoch $m * n$ Zustände [72], nämlich $q_{1,1}q_{2,1}, \ldots q_{1,1}q_{2,n}, q_{1,2}q_{2,1}, \ldots q_{1,2}q_{2,n}, q_{1,m}q_{2,1} \ldots q_{1,m}q_{2,n}$. Ein kombinierter Automat wird daher auch als Produktautomat bezeichnet [78].

Abstraktions- und Zuordnungsmodell EA haben kein Abstraktions- und Zuordnungsmodell [152]. Sie werden im Allgemeinen wegen ihrer formalen Beschreibung für die Verifikation von Steuerungsspezifikationen eingesetzt [175]. Für die meisten praktischen Anwendungsfälle sind die entstehenden Automaten zu komplex, daher ist eine Wartung bei Änderungen und Fehlern nur schwer möglich. Zudem werden EA von heutigen IEC 61131-3-kompatiblen Steuerungssystemen [23] nicht unterstützt, auch wenn automatische Transformationen in Strukturierten Text (ST) möglich sind [62].

Aktions- und Aktivitätenmodell Klassische EA nach [78] haben keine Möglichkeit, Aktionen oder Aktivitäten auszuführen. Sie reagieren nur durch die Übergangsfunktion in Abhängigkeit des aktuellen Zustands auf die Eingangssymbole. Die Bedingungen der Zustandsübergänge müssen also aus der Menge Σ stammen. Solche EA stellen Akzeptoren für eine formale Sprache dar [56].

Erst die Erweiterungen nach Mealy [108] und Moore [114] führen durch die Definition eines Ausgabealphabets ein einfaches Aktionsmodell ein. Solche EA werden als Transduktoren bezeichnet [56]. Für Aktionen wird die Automatendefinition um eine Ausgabefunktion $\lambda : Q \times \Sigma \rightarrow Y$ erweitert, wobei Y die Menge der Ausgänge ist [6]. Das Setzen von Ausgangsgrößen kann auf zwei verschiedene Weisen geschehen (vgl. Abbildung 3.3). Bei einem Mealy-Automat werden die Ausgangsgrößen während des Zustandsübergangs geschrieben, in einem Moore-Automat bei der Aktivierung des Zustands. Beide Typen lassen sich ineinander umformen [196]. Die Syntax der Aktionen ist durch die Ausgabefunktion festgelegt [56].

Ausführungssteuerungsmodell Auch das Ausführungssteuerungsmodell der DEA[4] ist sehr simpel konstruiert. Bei der Aktivierung eines Automaten wird der Startzustand aktiviert. Ein angelegtes Eingabewort wird Zeichen für Zeichen ausgewertet und entsprechend der Übergangsfunktion wird der Zustand geändert. Die Übergangsfunktion nimmt den Zustand als Ausgangswert, der beim jeweiligen Zeichen aktiv ist und nicht den, der beim Anlegen des Worts aktiv war. Der DEA terminiert, wenn er einen Endzustand erreicht hat [78].

Trifft ein DEA auf ein Zeichen, für das im aktuellen Zustand keine Übergangsfunktion existiert, ist der Automat in einem undefinierten Zustand. Daher müssen Eingaben, die der DEA ignorieren soll, explizit modelliert werden. Zudem kann es zu nicht erreichbaren Zuständen kommen, wenn keine Übergangsfunktion in einen Zustand hineinführt. Der DEA

[4] NEA lassen sich, wie bereits erwähnt, in einen DEA umformen und werden daher hier nicht betrachtet.

kann verklemmen, wenn aus einem Zustand, der kein Endzustand ist, keine Übergangsfunktion hinausführt [78].

3.3.2. Petrinetze

Petrinetze (PN) sind in den 1960er Jahren zur Modellierung allgemeiner dynamischer Strukturen entwickelt worden [175]. Wesentliche Arbeiten zu Petrinetzen sind durch Abel [1] und Schnieder [147] durchgeführt worden. PN bilden eine mathematische Sprache zur Systemmodellierung [165] und bauen auf den EA auf [196]. Der grundlegende Unterschied zwischen EA und PN liegt in der Zustandsdefinition. Ein EA befindet sich immer in genau einem Zustand, während bei einem PN der Zustand durch die Markenverteilung in allen Stellen definiert ist. Die mathematischen Grundlagen ermöglichen eine tiefgehende Analyse der erstellten Modelle [165]. In Abbildung 3.4 ist ein Beispiel für ein PN dargestellt.

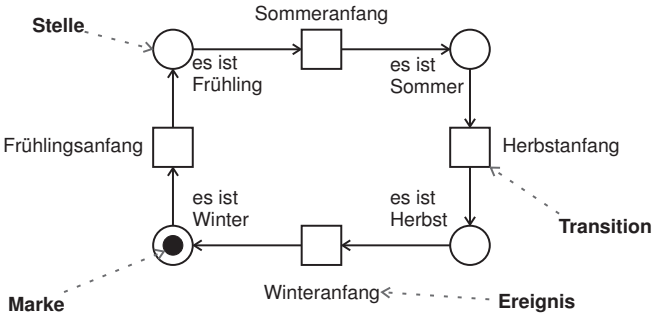

Abbildung 3.4.: Elemente eines Petrinetzes (nach [142])

Aufbaumodell Es wird eine ganze Klasse von Modellen als PN bezeichnet, eine Übersicht gibt z. B. [62]. Gemein ist all diesen PN-Varianten, dass sie aus Stellen, Transitionen und Kanten bestehen. Exemplarisch werden an dieser Stelle zwei verschiedene Definitionen vorgestellt:

Definition 1 Ein PN ist nach [141] definiert als ein 4-Tupel $(P, T, \mathbb{F}, \mathbb{B})$. Hierbei ist

- $P = \left\{p_1, \ldots, p_{|P|}\right\}$ eine endliche, angeordnete Menge von Stellen,
- $T = \left\{t_1, \ldots, t_{|T|}\right\}$ eine endliche, angeordnete Menge von Transitionen,
- $\mathbb{F}$ eine $|P| \times |T|$-Matrix über $\mathbb{N}$ und
- $\mathbb{B}$ eine $|P| \times |T|$-Matrix über $\mathbb{N}$.

Bei PN gilt $P \cap T = \emptyset$.

Definition 2 Ein PN nach [1] ist definiert als ein 6-Tupel (S, T, F, K, W, M_0). Hierbei ist

- $S = \left\{s_1, \ldots, s_{|P|}\right\}$ eine endliche, nichtleere Menge von Stellen,
- $T = \left\{t_1, \ldots, t_{|T|}\right\}$ eine endliche, nichtleere Menge von Transitionen,
- $F \subseteq S \times T \cup T \times S$ eine nichtleere Kantenmenge,
- $K : S \to \mathbb{N}$ die Abbildung, die die Marken-Kapazität einer Stelle definiert,
- $W : F \to \mathbb{N}$ die Abbildung, die den Kanten ihr Gewicht zuordnet und
- $M_0 : S \to \mathbb{N}$ die Anfangsmarkierung.

Die Menge $S \cup T$ stellt graphentheoretisch die Knotenmenge des Graphen dar, wobei auch hier $S \cap T = \emptyset$ gilt.

Die beiden Definitionen unterscheiden sich durch die Einführung der Marken in Definition 2. Die Marken bilden die Grundlage für die dynamischen Effekte eines PN, daher wird Definition 2 zur weiteren Analyse verwendet.

Hierarchie- und Vernetzungsmodell Stellen und Transitionen müssen alternierend miteinander verknüpft sein, d. h., ein PN ist ein bipartiter Graph. In Definition 1 enthält die Matrix $\mathbb{F}$ die Verbindungen zwischen Stellen und Transitionen, die Matrix $\mathbb{B}$ die Verbindungen zwischen Transitionen und Stellen. Die Einträge in $\mathbb{F}$ und $\mathbb{B}$ geben die Kantengewichte an [141]. In Definition 2 sind die Kanten in der Menge F enthalten. Die Kantengewichte werden durch die Abbildung W induziert [1]. Des Weiteren ist es notwendig, den Vorbereich und den Nachbereich eines Knotens zu definieren. Sei $x \in S \cup T$. Dann ist

- $\bullet x = \{y | (y, x) \in F\}$ der Vorbereich und
- $x \bullet = \{y | (x, y) \in F\}$ der Nachbereich

des Knoten x [116].

PN können in Teilnetze zerlegt werden. Allerdings sind nicht alle Eigenschaften des Gesamtnetzes aus den Teilnetzen ableitbar. Eine hierarchische Gliederung ist bei klassischen PN nicht möglich [98]. PN sind für die Modellierung nebenläufiger Prozeduren geeignet. Es können mehrere Stellen aktiv sein und das Schalten von Transitionen ist unabhängig von dem Status anderer Stellen. Die unabhängige Schaltbarkeit zweier Transitionen hat die Bedingung, dass die beiden Transitionen keine gemeinsamen Stellen in ihrem Vor- bzw. Nachbereich haben. Für eine bessere Übersichtlichkeit sind Elemente zum Eröffnen und zur Synchronisation der Parallelität eingeführt worden.

Abstraktions- und Zuordnungsmodell PN stellen lediglich eine abstrakte mathematische Basis dar. Erst durch Hinzufügen einer Semantik entstehen problemspezifische Lösungen. Dies wird als Interpretation bezeichnet. Steuerungstechnisch interpretierbare Petrinetze (SIPN) sind die für die Prozedurmodellierung wichtige Interpretation der PN. Hierbei entsprechen die Stellen Prozesseingriffen über Stellglieder. Die Transitionen reagieren auf Sensorwerte [98].

Aktions- und Aktivitätenmodell Die SIPN ermöglichen es, dass neben der Schaltbarkeit auch externe Gegebenheiten das Schalten von Transitionen beeinflussen können. Zudem verändern die Stellen die Ausgangssignale des SIPN. Hierzu werden zwei Funktionen $q_T(t)$ und $q_P(p)$ definiert. Die Funktion $q_T(t)$ wird als Schaltausdruck der Transition $t \in T$ und die Funktion $q_P(p)$ als Ausgabe der Stelle $p \in S$ bezeichnet. In Abbildung 3.5 ist ein Beispiel für SIPN abgebildet, welches die Eingänge x_1, x_2 und x_3 sowie die Ausgänge y_1 und y_2 besitzt.

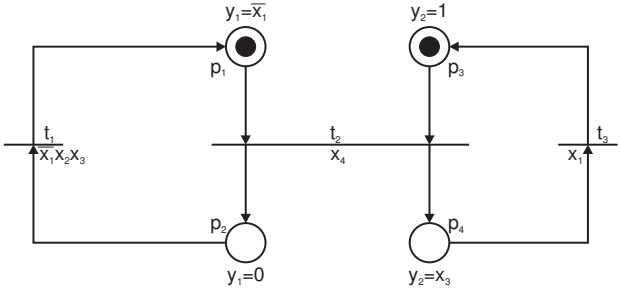

Abbildung 3.5.: Beispiel für ein SIPN (nach [98])

Das SIPN in Abbildung 3.5 besitzt die Schaltausdrücke

$$q_T \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} \overline{x_1} x_2 x_3 \\ x_4 \\ x_1 \end{pmatrix}$$

und die Ausgaben

$$q_P \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} (y_1 = \overline{x_1}, y_2 = -) \\ (y_1 = 0, y_2 = -) \\ (y_1 = -, y_2 = 1) \\ (y_1 = -, y_2 = x_3) \end{pmatrix}.$$

Ausführungssteuerungsmodell Neben den Kapazitäten der Stellen und der Anfangsmarkierung sind die Flussregeln für die Marken Grundlage für dynamische Vorgänge. Die Marken, auch Token genannt, sind zu Beginn auf die Stellen verteilt. Ausgehend von dieser Anfangsverteilung M_0 schalten die Transitionen [116]. Dieses Schalten wird auch als Feuern bezeichnet [141]. Damit eine Transition $t \in T$ feuern kann, müssen zwei Schaltregeln erfüllt sein [116]:

- $M(s) \geq W(s,t) \forall s \in {\bullet t}$ und
- $M(s) \leq K(s) - W(s,t) \forall s \in t{\bullet}$.

Voraussetzung ist demnach, dass so viele Marken im Vorbereich der Transition sind, dass die Kantengewichte erfüllt werden können. Zudem muss im Nachbereich der Transition genügend Kapazität für die weitergegebenen Marken vorhanden sein [116].

Ein PN hat kein globales Scheduling. Es gibt keine aktiven Schritte. Transitionen können jederzeit reagieren, wenn die Schaltregeln erfüllt sind. Der Zustand eines PN ist durch seine aktuelle Markierung gegeben [196]. Soll ein PN eine Zustandsmaschine darstellen, müssen alle Stellenkapazitäten und Kantengewichte den Wert Eins haben [142]. Bei bestimmten Konstruktionen ist es möglich, dass PN verklemmen, d. h., es kann keine weitere Transition schalten. Durch mathematische Analysen lässt sich jedoch feststellen, dass ein PN verklemmungsfrei ist.

3.3.3. Grafcet

Grafcet[5] ist eine Spezifikationssprache, die in den 1970er Jahren in Frankreich als Weiterentwicklung der PN entworfen worden ist [158]. Eine Ergänzung der PN um ein Konzept für die Einbettung in die Umgebung sowie um ein Hierarchiekonzept ist erfolgt, so dass sie SIPN ähneln [6, 158]. Grafcet ist in der IEC 60484 definiert. Ein Beispiel für einen Grafcet-Plan ist in Abbildung 3.6 abgebildet.

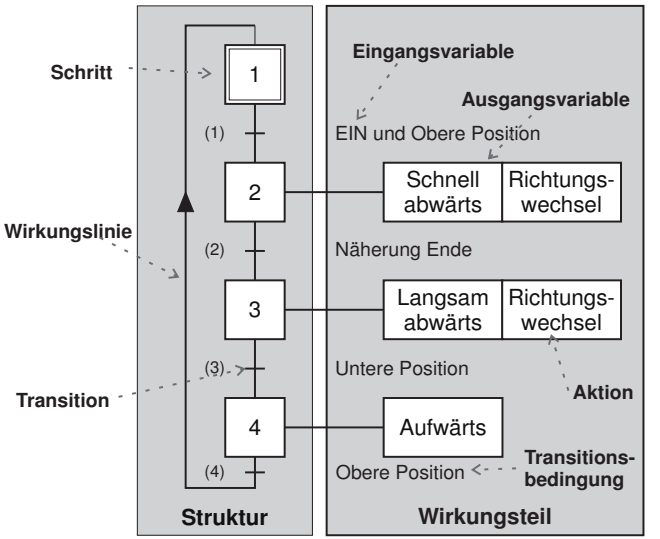

Abbildung 3.6.: Elemente eines Grafcet (nach [21])

Aufbaumodell Jeder Grafcet-Plan besteht zunächst aus zwei Teilen, der Struktur und dem Wirkungsteil (graue Rechtecke in Abbildung 3.6). Im Strukturteil sind die Entwicklungen zwischen verschiedenen Situationen durch Schritte, Transitionen und Wirkungslinien dargestellt. Es können ein oder mehrere Schritte als Anfangsschritt ausgezeichnet

[5] Die Bezeichnung „Grafcet“ ist eigentlich ein Akronym und steht für „GRAphe Fonctionnel de Commande Etapes/Transitions“

werden. Ein Schritt ohne vorangehende Transition ist ein Quellschritt und muss zwangsläufig Anfangsschritt sein. Schlussschritte hingegen sind optional. Der Wirkungsteil stellt die Beziehung zwischen den Ein- und Ausgangsvariablen und dem Strukturteil her. Transitionsbedingungen verarbeiten die aktuellen Eingänge, während Aktionen die Ausgangsvariablen setzen [21].

Hierarchie- und Vernetzungsmodell Die Schritte und Transitionen werden über Wirkbeziehungen verbunden [156]. Schritte und Transitionen sind alternierend verknüpft, so dass ein Grafcet-Plan ein bipartiter Graph ist. Alle Transitionen, von denen ein Schritt erreicht werden kann, bilden den Vorbereich des Schritts und alle Transitionen, aus denen ein Schritt verlassen werden kann, den Nachbereich eines Schritts [158]. Auf diese Weise erfolgt die Erzeugung von Ablaufketten mit Start und Ende sowie die von geschlossenen Ablaufketten [21].

Eine Strukturierung wird durch Einschließungen oder durch Makroschritte vorgenommen. Eine Einschließung stellt eine Möglichkeit dar, einen eigenständig entworfenen Grafcet als Ganzes in einen anderen Grafcet einzubinden. Ein Makroschritt hingegen ist ein Platzhalter für einen bestimmten Ausschnitt eines Grafcets [158]. Zwangssteuernde Befehle steuern Einschließungen und haben immer Vorrang vor den normalen Ablaufregeln [21]. Ein Grafcet-Plan kann sowohl Parallel- als auch Alternativverzweigungen enthalten [158]. Eine Alternative zu Parallelverzweigungen ist die Modellierung nebenläufiger Strukturen durch mehr als einen Initialzustand [6]. Die Alternativverzweigung wird als Ablaufauswahl bezeichnet und kann als Sonderfall das Überspringen von Schritten oder einen Rückführsprung modellieren [21].

In vielen Grafcet-Plänen sind Elemente wie Makroschritte, zwangssteuernde Befehle, einschließende Schritte oder zeitabhängige Bedingungen nicht enthalten [158]. Daher wird Anfängern geraten, auf diese Elemente zu verzichten [21].

Abstraktions- und Zuordnungsmodell Grafcet hat kein eigenes Abstraktions- und Zuordnungsmodell. Grafcet ist als Spezifikationssprache entwickelt worden, deren Implementation ein SFC sein kann [21]. Diese Transformation lässt sich sowohl auf Modell-Ebene als auch auf Meta-Modell-Ebene automatisiert durchführen [158]. Des Weiteren können einmal spezifizierte Grafcets durch Einschließungen wiederverwendet werden.

Aktions- und Aktivitätenmodell In einem Grafcet sind zwei Arten von Aktionen enthalten, diese sind kontinuierlich wirkende und gespeichert wirkende Aktionen. Eine kontinuierlich wirkende Aktion setzt den Wert einer Ausgangsvariable auf *wahr*, wenn der zugehörige Schritt aktiv und die Zuweisungsbedingung erfüllt ist. Ansonsten wird der Wert der Ausgangsvariable auf *falsch* gesetzt, wenn keine andere Aktion den Wert auf *wahr* setzt. Im Gegensatz zu kontinuierlich wirkenden Variablen bleibt das Ergebnis einer gespeichert wirkenden Aktion so lange bestehen, bis eine andere Aktion den Wert wieder ändert [21].

Transitionsbedingungen sind boolesche Ausdrücke, die sich aus den Eingangsvariablen, internen Variablen und internen Ereignissen zusammensetzen. Bei den Eingangsvariablen werden neben dem Wert einer binären Variable auch Änderungen von binären Werten unterstützt. Es wird zwischen einer positiven Flanke (der Eingang x wechselt von *falsch* auf *wahr*, $\uparrow x$) und einer negativen Flanke (der Eingang x wechselt von *wahr* auf *falsch*, $\downarrow x$) unterschieden. In Transitionsbedingungen können auch Aussagen integriert sein, z. B.

Vergleiche von numerischen Werten. Als interne Ereignisse werden eine Schrittaktivierung, eine Schrittdeaktivierung oder das Auslösen einer Transition genannt [21].

Ausführungssteuerungsmodell Die Anfangssituation wird durch die Anfangszustände festgelegt. Eine Änderung des Zustands ergibt sich durch das Feuern von Transitionen. Eine Transition feuert, wenn die Transition freigegeben und die Transitionsbedingung *wahr* ist. Eine Transition ist freigegeben, wenn alle Schritte vor der Transition aktiv sind. Es können mehrere Transitionen gleichzeitig feuern. Feuert eine Transition, werden alle vorangegangenen Schritte deaktiviert und alle folgenden Schritte aktiviert. Wird ein Schritt gleichzeitig durch eine Transition deaktiviert und durch eine andere aktiviert, bleibt er aktiv.

In einem Grafcet können transiente und nichttransiente Abläufe auftreten. Bei einem transienten Ablauf wird ein Schritt aktiviert und direkt wieder deaktiviert, da eine seiner ausgehenden Transitionen ebenfalls feuert. Dies führt dazu, dass in diesem Schritt nur Zuordnungen in gespeichert wirkenden Aktionen ausgeführt werden, Zuweisungen in kontinuierlich wirkenden Aktionen jedoch nicht.

Konflikte können nur bei gespeichert wirkenden Aktionen vorkommen, bei kontinuierlich wirkenden Aktionen ist im Konfliktfall das Signal *wahr* [158]. Die Verhinderung der Konfliktfälle ist Aufgabe des Entwicklers [21].

3.3.4. Procedural Function Charts

PFC werden in der IEC 61512-2[6] [36] als Ableitung der SFC zur Beschreibung von Steuerungsprozeduren für Batch-Prozesse definiert [175]. Sie sind eng mit den Modellen zur Strukturierung von Batch-Prozessen (Prozessmodell) und Batch-Anlagen (Physisches Modell) aus [35] verknüpft [67]. Prozeduren sind neben dem Rezeptkopf, Stoff- und Produktionsdaten sowie Anforderungen an die Einrichtung Bestandteil eines Rezepts [35]. Die wesentlichen Elemente eines PFC sind in Abbildung 3.7 dargestellt.

Aufbaumodell Die Logik der Prozedur wird in einem PFC durch eine Reihe von Symbolen dargestellt. Die Rezept-Prozedurelemente entsprechen den Schritten eines Grafcets. PFC müssen im Gegensatz zu SFC mindestens einen Anfangs- und mindestens einen Endpunkt haben. Ein Schritt kann speichernd und nicht-speichernd sein [37]. Belegungssymbole dienen der Zuordnung von Betriebsmitteln (oder Apparaten) zu den einzelnen Rezept-Prozedurelementen. Über Element-Synchronisationen sind zwei Rezept-Prozedurelemente verknüpft, die synchron ausgeführt werden sollen. Diese Synchronisation kann einen Materialtransport beinhalten. Zwischen zwei Rezept-Prozedurelementen befindet sich eine Transition, die in explizite und implizite Transitionen differenziert wird [36].

Hierarchie- und Vernetzungsmodell Die Rezept-Prozedurelemente sind über Transitionen miteinander verbunden. PFC ermöglichen nur Abläufe mit Anfang und Ende, zyklische Abläufe sind nicht erlaubt. Neben linearen Ketten sind Ablauf-Selektionen und parallele Abläufe in der Sprache enthalten. Ablauf-Selektionen müssen explizite Transitionen haben,

[6] Die IEC 61512 basiert auf der ISA 88, die von der International Society of Automation (ISA) veröffentlicht wurde.

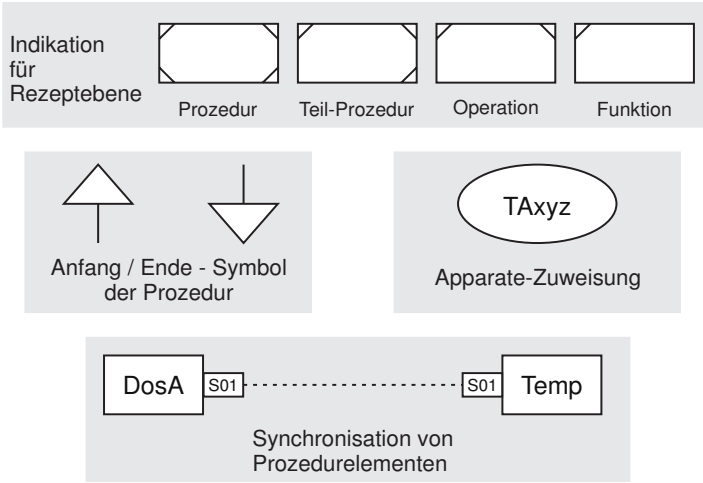

Abbildung 3.7.: Basiselemente eines PFC (nach [63])

da ansonsten keine Selektion möglich ist. Parallele Abläufe müssen vor dem Endpunkt zu einem Pfad zusammengeführt werden [36].

PFC können hierarchisch strukturiert werden. Die Rezept-Prozedurelemente umfassen vier verschiedene Ebenen: die Prozedur, die Teilprozedur, die Operation und die technische Funktion (vgl. Abbildung 3.7). Prozeduren beschreiben die Steuerung einer vollständigen Verarbeitungsfunktion, z. B. die Herstellung einer Charge. Teil-Prozeduren steuern eine Produktionssequenz, die auf genau einer Teilanlage abläuft. Operationen kontrollieren Prozesse, die den Zustand von Stoffen verändern. Funktionen greifen direkt auf die Einrichtungssteuerung (vgl. Aktions- und Aktivitätenmodell) zu. Jedes Rezept-Prozedurelement mit Ausnahme einer Funktion kann aus Rezept-Prozedurelementen der tieferen Ebene zusammengesetzt werden [35].

Abstraktions- und Zuordnungsmodell Das Abstraktions- und Zuordnungsmodell ergibt sich durch das Rezeptmodell aus [35]. Das Rezeptmodell umfasst vier Rezepttypen, Verfahrensrezepte, Werksrezepte, Grundrezepte und Steuerrezepte. Beschreibungsmittel für alle vier Rezepttypen sind PFC. Verfahrens- und Werkrezepte stellen das prinzipielle Vorgehen in den Vordergrund, während Grund- und Steuerrezepte sich auf die tatsächlichen Betriebsmittel beziehen. Verfahrensrezepte werden auf Unternehmensebene entwickelt und enthalten die abstrakten Steuerungsschritte, die für den Prozess zur Erzeugung des gewünschten Produkts notwendig sind. Werksrezepte enthalten zusätzlich Informationen aus dem lokalen Kontext des Produktionsorts, z. B. Sprachinformationen oder rechtliche Rahmenbedingungen. Erst mit dem Grundrezept wird der Bezug zu den Einrichtungen der Anlage hergestellt. Das Steuerrezept ist eine Kopie des Grundrezepts, das zusätzliche Informationen für eine bestimmte Charge enthält [35]. Des Weiteren motiviert die IEC 61512-3

eine Bibliothek für wesentliche Ablaufereignisse [37].

Aktions- und Aktivitätenmodell Steuerrezepte können nur in Kombination mit der Einrichtungssteuerung einen Prozess steuern. Diese Einrichtungssteuerung umfasst die Ebenen 1 und 2 eines PLS (vgl. Kapitel 2.4, S. 14) und ist nicht Teil des Rezeptmodells. Die Verknüpfung zwischen dem Steuerrezept und dem PLS wird durch Einrichtungs-Prozedurelemente hergestellt. Diese Verknüpfungen werden als Belegungssymbole bezeichnet. Dies geschieht über Verweise im Steuerrezept [35], die in der IEC 61512-2 definiert sind.

Die Aktionen in einem speichernden Schritt bleiben bis zur Überschreibung in einem anderen Schritt aktiv. Aktionen in nicht-speichernden Schritten sind nur dann aktiv, wenn der jeweilige Schritt aktiv ist [37]. Explizite Transitionen besitzen eine Transitionsbedingung, implizite Transitionen nicht. Es ist keine Sprache festgelegt, in der Transitionsbedingungen beschrieben werden sollen [36].

Ausführungssteuerungsmodell Ein PFC wird durch den zugehörigen Zustandsautomaten[7] (vgl. Abbildung 3.8) gestartet.

Dies führt dazu, dass der auf den Anfangspunkt folgende Schritt aktiviert wird [36]. Das Weiterschalten erfolgt mittels Transitionen, wobei implizite Transitionen direkt feuern, sobald der vorherige Schritt beendet ist. Explizite Transitionen fordern den vorherige Schritt auf, sich zu beenden, sobald ihre Transitionsbedingung *wahr* ist. Sind in einer Ablauf-Selektion mehrere Transitionsbedingungen wahr, erfolgt die Auswertung der Transitionen von links nach rechts [36]. Im Allgemeinen sind die Schritte im PFC jedoch selbstbeendend und schalten durch implizite Transitionen weiter [175]. Die Ausführung eines PFC kann durch andere PFC beeinflusst werden, wenn diese über Synchronisationen verknüpft sind [36].

Der Zustandsautomat (vgl. Abbildung 3.8) realisiert ebenfalls die Ausnahmebehandlung während der Prozedurausführung. Hierbei sind vier unterschiedliche Reaktionen der steuernden Prozedur auf ein unerwartetes Ereignis möglich: Unterbrechen, Anhalten, Stoppen und Abbrechen [35]. Die vier Reaktionen werden nach dem Grad des Ereignisses ausgewählt. Nach einer Unterbrechung der Prozedurausführung kann die Prozedur an derselben Stelle wieder fortgesetzt werden. Anhalten bedeutet, dass die Ausführung der Prozedur nicht ohne zusätzliche Aktionen möglich ist. Eine gestoppte Prozedur führt den Prozess in einen sicheren Zustand. Anschließend ist ein expliziter Neustart erforderlich. Ein Abbruch bringt den Prozess schnellstmöglich in einen sicheren Zustand und nimmt dabei Schäden an der Anlage in Kauf [62]. Allerdings ist in der IEC 61512-1 nicht definiert, wie sich unterlagerte Ebenen verhalten müssen, wenn eine höhere Ebene in eine Ausnahme läuft [137].

3.3.5. Grafchart

Neben dem SFC ist auch Grafchart eine auf Grafcet basierende Programmiersprache. Grafchart ist eine Ergänzung von Grafcet um Programmierhochsprachen und Objektorientierung. Des Weiteren fließen verschiedene Konzepte der Petrinetze ein. Die Entwicklung von

[7] Die Grafik basiert auf der Zustandsübergangsmatix in [35] und ist an die Version im Draft 2c vom Januar 2014 angelehnt.

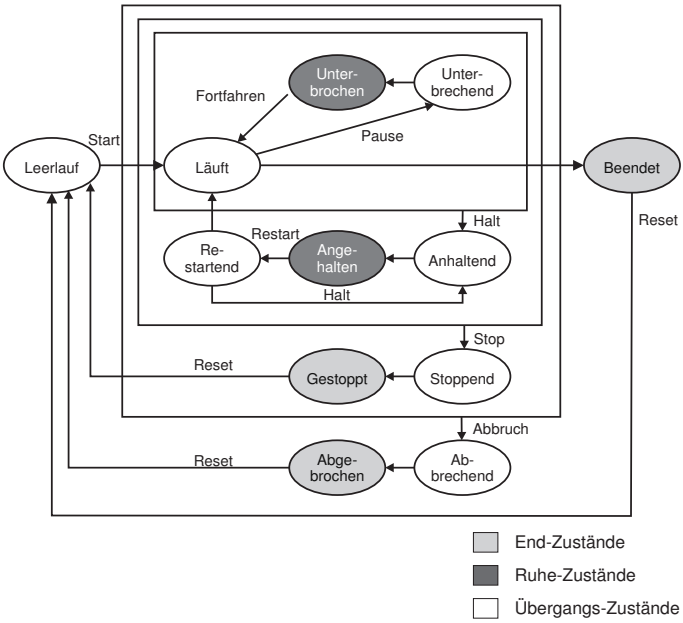

Abbildung 3.8.: Zustands-Übergangsdiagramm für beispielhafte Zustände von Prozedur-Elementen (nach [35])

Grafchart hat zwei Ziele verfolgt, zum einen das Aufzeigen der Entwicklung einer objektorientierten grafischen Programmiersprache ausgehend von einer grafischen Spezifikationssprache, zum anderen sollte die bestehenden Analysemethoden für PN weiter anwendbar sein [89]. Es gibt zwei Implementierungen von Grafchart, eine in der Programmiersprache G2, eine in der Programmiersprache Java [165].

Grafchart wird zur Steuerung von Batch-Prozessen verwendet [88]. Aber auch in der Fertigungsautomation findet Grafchart in universitären Anwendungen Verwendung [136, 165]. In Abbildung 3.9 sind die typischen Elemente von Grafchart aufgeführt.

Aufbaumodell Ein Grafchart besteht analog zu Grafcet und SFC primär aus Schritten und Transitionen. Die Schritte repräsentieren Zustände, die Transitionen die Änderung von Zuständen. Ein Schritt in Grafchart hat drei Attribute, x, t und s. Das Attribut x gibt an, ob der Schritt aktiv ist oder nicht. Die Attribute s und t beinhalten die Information, wie lange der Schritt aktiv ist, s beinhaltet die Dauer in Sekunden, t die Dauer in SPS-Zyklen. Anfangs- und Endschritte sind ausgezeichnete Schritte. Aus Gründen der Übersichtlichkeit können Verbindungspunkte eingefügt werden [5]. Transitionen dienen als Verbindungsobjekte zwischen Schritten. Hierbei existieren spezielle Exception-Transitionen, die das Verlassen eines Makroschritts oder einer Prozedur im Fehlerfall darstellen. Des Weiteren gibt

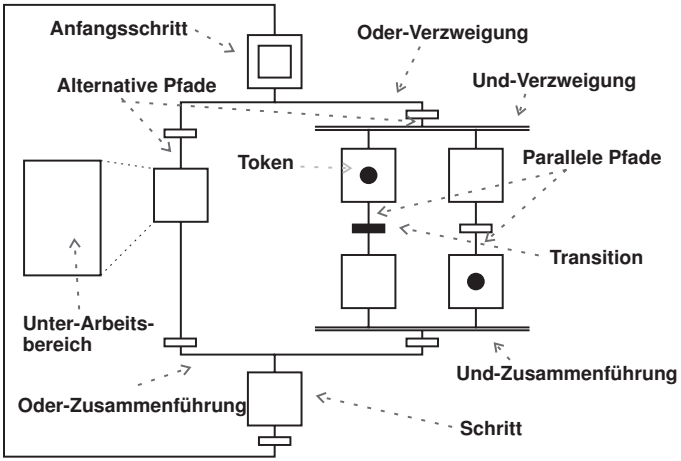

Abbildung 3.9.: Elemente von Grafchart (nach [88])

es in Grafchart Variablen [165].

Hierarchie- und Vernetzungsmodell Schritte und Transitionen sind in Grafchart alternierend verknüpft. Grafchart unterstützt alternative und parallele Pfade. Jede Parallelverzweigung erzeugt zwei parallele Zweige und eine Vereinigung führt die Pfade wieder zusammen. Sind mehr als zwei parallele Pfade notwendig, müssen mehrere Verzweigungen bzw. Zusammenführungen hintereinander geschaltet werden [165].

Grafchart unterstützt Makroschritte und Prozeduren als Hierarchieelemente. Makroschritte fassen, wie in Grafcet, mehrere Schritte zu einem Element zusammen und erhöhen die Übersichtlichkeit. Prozeduren können zusätzlich Parameter übergeben bekommen und Rückgabewerte liefern. Hierbei wird zwischen Call-by-reference und Call-by-value[8] unterschieden [165]. Prozeduren können durch Prozedurschritte und Prozessschritte aufgerufen werden. Während bei einem Prozedurschritt die Ausführung der aufgerufenen Prozedur im selben Thread erfolgt, erzeugt ein Prozessschritt einen neuen Thread [88].

Abstraktions- und Zuordnungsmodell Prozeduren in Grafchart lassen sich wiederverwenden. Auf diese Weise kann die Erzeugung von doppeltem Code vermieden werden. Durch die Parameter sind Prozeduren auch flexibel an verschiedene Rahmenbedingungen anpassbar. Ihr interner Aufbau bleibt allerdings starr. Denkbar sind jedoch Alternativpfade, die über einen Parameter steuerbar sind [165].

Aktions- und Aktivitätenmodell Aktionen sind Schritten zugeordnet und im Unterarbeitsbereich eines Schritts enthalten [88]. In der ursprünglichen Version sind Aktionen in

[8] Zur Erkläung von Call-by-value und Call-by-reference siehe z. B. [127].

den Regeln der Programmiersprache G2 definiert [5]. Transitionen besitzen eine Wächterbedingung, die aus booleschen Ausdrücken oder Ereignissen zusammengesetzt sein kann [165].

Aktionen in Grafchart besitzen ein Präfix, welches die Ausführung der Aktion kontrolliert. In Tabelle 3.1 ist eine Übersicht der möglichen Präfixe gegeben [165].

Tabelle 3.1.: Liste der möglichen Grafchart-Präfixe [165]

Präfix	Beschreibung
S	Die Aktion wird ausgeführt, wenn der Schritt aktiviert wird.
X	Die Aktion wird ausgeführt, wenn der Schritt deaktiviert wird.
P	Die Aktion wird zyklisch ausgeführt, solange der Schritt aktiv ist.
N	Die Variable ist mit dem Zustand des Schritts assoziiert, d. h., sie ist *wahr*, wenn der Schritt aktiv ist und ansonsten *falsch*.
A	Die Aktion wird ausgeführt, wenn der Schritt abgebrochen wird.
V	Prozedurparameter (Call-by-value)
R	Prozedurparameter (Call-by-reference)

Neben den klassischen Aktionen bietet Grafchart auch den Aufruf von Diensten an. Auf diese Weise kann Grafchart auch in eine SOA integriert werden. Realisiert wird dies durch die Einbindung von Device Profiles for Web Services (DPWS) in Grafchart [136, 167]. Ferner ist auch eine OPC UA-Anbindung bereits in Grafchart implementiert [166].

Ausführungssteuerungsmodell Ein Grafchart wird wie ein SFC zyklisch ausgeführt. Ein Ausführungsmodell, welches in jedem Zyklus die folgenden Operationen ausführt [122], steuert die Ausführung eines Grafcharts.

1. Die digitalen und analogen Eingänge werden gelesen.
2. Aktivierte Transitionen mit wahrer Bedingung werden als feuerbar markiert.
3. Bei Konflikten wird die Markierung bei Transitionen mit niedrigerer Priorität wieder entfernt. Exception-Transitionen haben gegenüber normalen Transitionen Priorität.
4. Die feuerbaren Transitionen feuern, während sie zwei Operationen ausführen:
   a) Schritte vor der Transition werden deaktiviert. Dabei werden insbesondere Aktionen mit Präfix X ausgeführt. Die Deaktivierung eines Schritts deaktiviert alle nachfolgenden Transitionen [5].
   b) Schritte nach der Transition werden aktiviert. Dabei werden insbesondere Aktionen mit Präfix S ausgeführt. Die Aktivierung eines Schritts aktiviert alle nachfolgenden Transitionen [5].
5. Für jeden Schritt werden drei Operationen ausgeführt:
   a) Die Attribute t und s des Schritts werden aktualisiert.
   b) Aktionen mit dem Präfix P werden ausgeführt, wenn der Schritt aktiv ist und im vorherigen Zyklus aktiv war.

c) Variablen, die mit Aktionen mit dem Präfix N verknüpft sind, werden geändert, wenn sich der Zustand des Schritts seit dem letzten Zyklus verändert hat.

6. Der Grafchart pausiert bis zum nächsten Zyklus.

Durch dieses Vorgehen ist ein vollständig deterministisches Ausführungsverhalten möglich. Die Auswertereihenfolge der Transitionen ist irrelevant und bei mehreren wahren Bedingungen einer Alternativverzweigung werden auch mehrere Zweige aktiviert [122]. Dies hat sich im Gegensatz zu früheren Versionen von Grafchart verändert. Dort ist in diesem Fall nichtdeterministisch eine Wahl getroffen worden. In Grafchart ist zudem die Ausführung aller Aktionen mit Präfix S und X garantiert, da Schritte nicht durchschalten können [5].

Makroschritte und durch Prozedur-Schritte aufgerufene Prozeduren laufen bis zu ihrem Endschritt durch, bevor die aufrufende Kette weiterschalten kann. Wie bereits beschrieben, haben Exception-Transitionen jedoch gegenüber normalen Transitionen Priorität und können nur an Makroschritten und Prozeduren angebunden sein. Feuert eine solche Exception-Transition während der Ausführung eines Makroschritts bzw. einer Prozedur, wird der Makroschritt bzw. die Prozedur abgebrochen und die Ausführungssteuerung führt Aktionen mit Präfix A aus. Bei einer erneuten Aktivierung des Makroschritts bzw. der Prozedur startet die interne Logik wieder an der Stelle, an der der Abbruch erfolgt ist [137].

Der Aufruf einer Prozedur durch einen Prozessschritt führt zur Erzeugung einer zweiten Ausführungssteuerung. Dies hat zur Konsequenz, dass die Ausführung der aufgerufenen Prozedur unabhängig von der aufrufenden Prozedur erfolgt. Abhängigkeiten zwischen den Prozeduren müssen explizit durch Aktionen und Transitionsbedingungen projektiert sein [137].

3.3.6. Sequential Function Charts

SFC sind als eine mögliche Implementierung von Grafcet in der IEC 61131-3 [23] definiert. Sie sind eine weit verbreitete Darstellung der kausalen Struktur von Prozeduren in Steuerungssystemen [175]. Auch die IEC 61499-1 [26] nennt SFC als geeignetes Mittel zur Beschreibung von Prozeduren. SFC lassen sich als Graph und als Zustandsübergangsmatrix darstellen [133]. Neben der ursprünglichen Anwendung als Programmiersprache stellen SFC eine Alternative zu Grafcet als Spezifikationssprache[9] dar. Es wird eine identische grafische Repräsentation für Engineering und Laufzeit genutzt [3]. In Abbildung 3.10 sind die wesentlichen Elemente eines SFC abgebildet.

Aufbaumodell In einem SFC stellt der Schritt eine definierte Situation der Steuerung dar. Ein Schritt hat einen Schrittmerker zur Unterscheidung der Aktivität/Inaktivität und hält seine aktive Zeit als Variable zugriffsbereit vor. Transitionen sind Übergänge zwischen den Schritten [23]. In der IEC 61131-3 ist kein Endschritt definiert [196].

Hierarchie- und Vernetzungsmodell SFC sind in sogenannten Programm-Organisationseinheiten (POE) enthalten [62]. In einem SFC werden ausgehend vom Initialschritt Transitionen und Schritte alternierend verknüpft, d. h., ein SFC ist ein bipartiter Graph. Durch eine Alternativverzweigung ist die Auswahl zwischen mehreren Ketten möglich. Die

[9] Die Spezifikation ist eine Beschreibung des Verhaltens einer Steuerung, während sich die Implementierung auf eine konkrete Programmiersprache und auf eine spezifische Hardware bezieht [6].

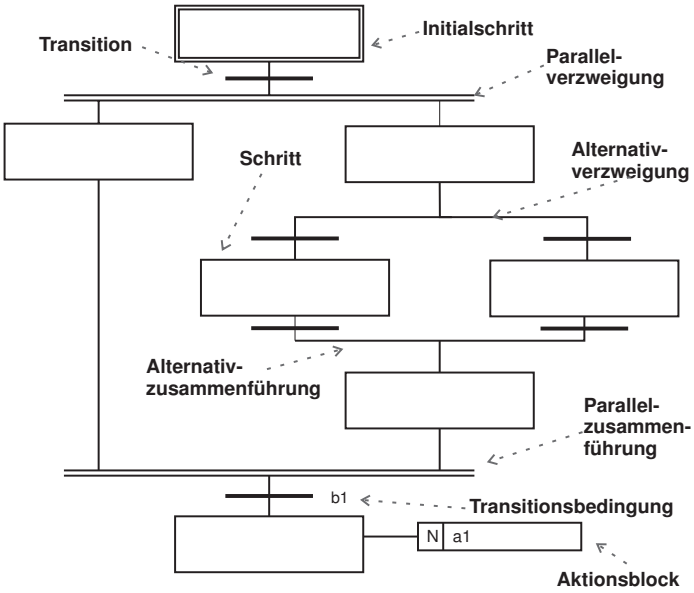

Abbildung 3.10.: Elemente eines SFC (nach [116])

Festlegung der Priorität der Transitionen kann hierbei auf drei Arten erfolgen, durch Vorrang von links nach rechts, durch Nummerierung und durch gegenseitigen Ausschluss. Der Kettensprung und die Kettenschleife sind Sonderfälle der Auswahlverzweigung. Simultanketten sind mit einer Parallelverzweigung ebenfalls möglich [23]. Die Parallelverzweigung verbindet eine Transition mit mindestens zwei Schritten, die entsprechende Zusammenführung verbindet mindestens zwei Schritte mit einer Transition [116].

Als Hierarchiemodell bietet sich die Möglichkeit des Aufrufs anderer SFC an [23]. Dies ist jedoch fehleranfällig. Die SFC können sich gegenseitig verklemmen, wenn jeder SFC auf eine Schrittänderung des jeweils anderen wartet [24]. Zudem ist dieser Aufruf in der IEC 61131-3 nicht vollständig spezifiziert. So ist das Verhalten des aufrufenden SFC nicht eindeutig beschrieben, wenn der aufgerufene SFC deaktiviert wird [156]. Deshalb ist diese Möglichkeit auch explizit im Beiblatt 1 zur IEC 61131-3[10] [24] als nicht empfohlen gekennzeichnet. Eine andere Möglichkeit besteht in der Integration von SFC in Funktionsbausteine. Dies erhöht zum einen die Wartbarkeit, zum anderen lassen sich die Funktionsbausteine wiederverwenden [24].

Abstraktions- und Zuordnungsmodell Im Beiblatt 1 ist ein Top-Down-Entwurfsprozess und eine Bottom-Up-Implementierung als Paradigma für die Erstellung von SFC vorge-

[10]Das Beiblatt bezieht sich auf die Vorgängerversion von [23]. Diese weicht nach [156] nur unwesentlich im Bereich der SFC von der aktuellen Version ab.

schlagen. Nach der Festlegung der Gesamtfunktionalität und der äußeren Schnittstelle wird die Funktionalität in immer kleinere Elemente unterteilt, bis entweder das Element in einer Bibliothek vorhanden ist oder durch die Sprachen der IEC 61131-3 algorithmisch ausgedrückt werden kann. Diese atomaren funktionalen Einheiten werden implementiert bzw. instantiiert und anschließend verschaltet [24]. Es ist unter bestimmten Rahmenbedingungen möglich, SFC in einen Funktionsbaustein in Funktionsbaustein-Sprache (FBS) umzuwandeln [191]. Eine Abstraktion der Funktionsbausteine ist mit dem Typ-Instanz-Konzept möglich.

Die Richtlinie ISA TR 106 [81] schlägt ein Vorgehen zum Engineering einer Prozedur vor. Hierbei formuliert der Entwickler Anforderungen an die Steuerungsprozedur basierend auf einem Prozessmodell, aus denen er im nächsten Schritt zusammen mit dem Anlagenmodell ein Implementierungsmodell erzeugt. In diesem Implementierungsmodell ist anders als in der IEC 61512 [35] ein direkter Zugriff auf die Einzelsteuerebene enthalten.

Aktions- und Aktivitätenmodell Aktionen können in SFC in zwei Weisen auftreten, nämlich zum einen als boolesche Variablen, zum anderen können sie in den anderen Programmiersprachen der IEC 61131-3 definiert sein. Hierzu zählen Anweisungen in Anweisungsliste (AWL) oder ST, Strompfade in Kontaktplan (KOP), Netzwerke in FBS oder insbesondere andere SFC. Jede Aktion besitzt ein Aktionsbestimmungszeichen, das für das Ausführungsmodell wichtig ist (vgl. Tabelle 3.2). Aktionen lassen sich zu Aktionsblöcken zusammenfassen, die den Schritten zugewiesen sind [23].

Transitionen besitzen eine Transitionsbedingung. Eine Transitionsbedingung kann als boolescher Ausdruck in ST formuliert sein, es können aber auch der Ausgang eines KOP bzw. eines FBS genutzt werden. Zudem ist es möglich, Konstrukte in ST oder AWL zu verwenden [23].

Ausführungssteuerungsmodell Die Ausführung eines SFC erfolgt innerhalb der POE. Sobald die POE initialisiert ist, wird auch der Initialschritt aktiviert. Der Schrittwechsel erfolgt durch das Schalten von Transitionen. Eine Transition schaltet, wenn die Transition freigegeben ist, d. h., wenn alle vorausgehenden Schritte aktiv sind und die Transitionsbedingung *wahr* ist [23]. Die Regel der PN, dass die Schritte nach der Transition genügend Platz für Marken haben müssen, existiert bei SFC nicht [6]. Wenn eine Transition feuert, werden die vorherigen Schritte deaktiviert und die nachfolgenden Schritte aktiviert. Der Schaltvorgang soll idealerweise keine Zeit in Anspruch nehmen, in der Praxis ist er aber von der SPS abhängig [23]. Die Ausführungssteuerung der Aktionen erfolgt durch einen internen Funktionsbaustein. Dieser interpretiert die zugehörigen Aktionsbestimmungszeichen der Aktionen und sorgt für die entsprechende Ausführung [23].

Die Ausführung von SFC erfolgt entweder nach dem Maximal-Progress-Vorgehen oder nach dem Lock-Step-Vorgehen. Bei dem Maximal-Progress-Vorgehen feuern zunächst alle Transitionen, bis ein stabiler Zustand erreicht ist. Anschließend werden die Aktionen der dann aktiven Schritte ausgeführt. Beim Lock-Step-Vorgehen feuern nur die Transitionen, die im selben Zyklus aktiviert worden sind [62]. Zudem unterscheiden sich Realisierungen von SFC in der Reihenfolge der Aktionsausführung und Transitionsauswertung. Je nachdem, ob zuerst Transitionen ausgewertet oder Aktionen ausgeführt werden, kann das Resultat ein anderes sein. Auch die Reihenfolge der Aktionsausführung in einem Schritt kann zu Konflikten führen [6]. Die meisten kommerziellen Realisierungen der SFC verwenden

Tabelle 3.2.: Liste der möglichen Aktionsbestimmungszeichen eines SFC (nach [23, 62])

Nummer	Zeichen	Beschreibung
1	Kein	Nicht gespeichert. Die zugehörige Aktion wird ausgeführt, während der Schritt aktiv ist.
2	N	Nicht gespeichert. Die zugehörige Aktion wird ausgeführt, während der Schritt aktiv ist.
3	R	Vorrangiges Rücksetzen. Die zugehörige Aktion wird nicht mehr ausgeführt.
4	S	Setzen (gespeichert). Die zugehörige Aktion wird ausgeführt, bis sie rückgesetzt wird.
5	L	Zeitbegrenzt. Die zugehörige Aktion wird ausgeführt, bis entweder die Zeitspanne abgelaufen ist oder der Schritt deaktiviert wird.
6	D	Zeitverzögert. Die zugehörige Aktion wird nach Ablauf der Zeitspanne ausgeführt, bis der Schritt deaktiviert wird.
7	P	Impuls (Flanke). Die zugehörige Aktion wird bei der Aktivierung und bei der Deaktivierung des Schritts einmal ausgeführt.
8	SD	Gespeichert und zeitverzögert. Die zugehörige Aktion wird nach Ablauf der Zeitspanne ausgeführt, bis sie rückgesetzt wird.
9	DS	Verzögert und gespeichert. Die zugehörige Aktion wird nach Ablauf der Zeitspanne ausgeführt, bis sie rückgesetzt wird, es sei denn, der Schritt wird vor Ablauf der Zeitspanne deaktiviert.
10	SL	Gespeichert und zeitbegrenzt. Die zugehörige Aktion wird ausgeführt, bis die Zeitspanne abgelaufen ist.
11	P1	Puls (steigende Flanke). Die zugehörige Aktion wird bei der Aktivierung des Schritts einmal ausgeführt.
12	P0	Puls (fallende Flanke). Die zugehörige Aktion wird bei der Deaktivierung des Schritts einmal ausgeführt.

das Lock-Step-Vorgehen in Kombination mit der Auswertung der Transitionsbedingungen vor der Aktionsausführung [196].

Während der Ausführung eines SFC stoppt bei einem Fehler normalerweise der Programmfluss. Anschließend wird entweder der Fehler automatisch korrigiert, falsche Variablenwerte durch ihren Default-Wert ersetzt oder auf eine manuelle Korrektur des Fehlers gewartet. Anschließend setzt sich der Ablauf an einer geeigneten Stelle fort [24].

Das Ausführungssteuerungsmodell der SFC schließt explizit die Erstellung unsicherer bzw. verklemmender Ablaufketten nicht aus [23]. Dieses Verhalten kann bei der Verwendung von geschachtelten Simultanverzweigungen oder bei einem Mischen von Simultan- und Alternativverzweigungen auftreten [24]. In diesem Fall kann sich die Anzahl aktiver Schritte unkontrolliert vermehren [6].

3.3.7. Zustandsdiagramme

Zustandsdiagramme, auf Englisch Statecharts (SC), sind eine Erweiterung von EA, die zur Modellierung reaktiver Systeme entwickelt worden sind [72]. Sie basieren auf der Idee, dass Systeme zu jeder Zeit einen definierten Zustand haben. Systeme besitzen unterschiedliche Zustände und reagieren in jedem dieser unterschiedliche Zustände in einer anderen Art und Weise [93]. Der Wechsel von einem Zustand zu einem anderen Zustand erfolgt durch externe und interne Ereignisse. Zusammengefasst ist das Verhalten reaktiver Systeme durch die Menge der Sequenzen von Eingangs- und Ausgangsereignissen, Bedingungen, Aktionen und zeitlichen Rahmenbedingungen bestimmt [72]. In Abbildung 3.11 ist ein Beispiel für einen SC dargestellt.

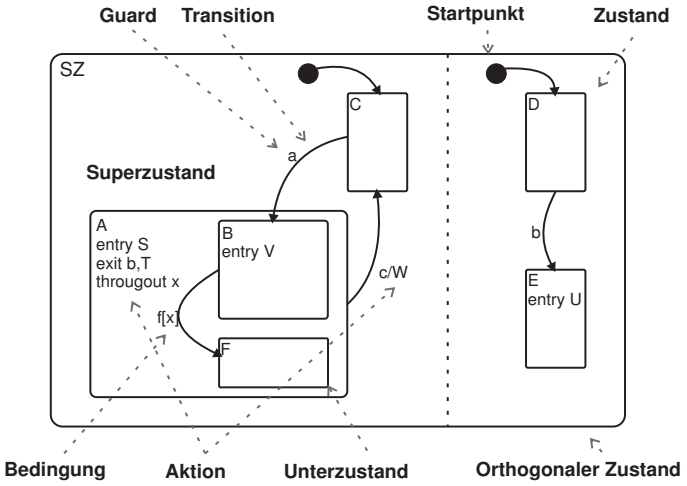

Abbildung 3.11.: Beispiel für einen SC (nach [72])

SC gehören zu den Verhaltensmodellen innerhalb der UML/SysML. Es ist eine modellbasierte Generation von Quellcode möglich, wodurch eine große Akzeptanz bei Software-Entwicklern entsteht [4].

Aufbaumodell Ein SC besteht aus Zuständen und Zustandsübergängen [72]. Jeder Zustand hat einen Namen und beschreibt eine statische oder dynamische Situation [93]. Zustandsübergänge haben ein zugeordnetes Ereignis und können optional Bedingungen besitzen [72]. Es gibt ausgezeichnete Start- und Endzustände, deren Verwendung zwingend ist. Ein Terminator ist ein spezieller Endzustand, der einen Abbruch des eigentlichen Ablaufs kennzeichnet [196].

Hierarchie- und Vernetzungsmodell Generell verbinden bei SC ähnlich wie bei EA Zustandsübergänge die Zustände. Zur Verhinderung der Zustandsexplosion, die bei den EA

auftritt, besitzen SC ein Hierarchiemodell [72]. Hierzu werden sogenannte Superzustände verwendet, die ihrerseits Unterzustände und interne Zustandsübergänge zwischen den Unterzuständen haben [11]. Zustandsübergänge können, wie in Abbildung 3.12 gezeigt, an einem Superzustand oder an einem Unterzustand in einem Superzustand beginnen bzw. enden [72]. Der zweite Fall wird auch als Inter-Level-Übergang bezeichnet [6].

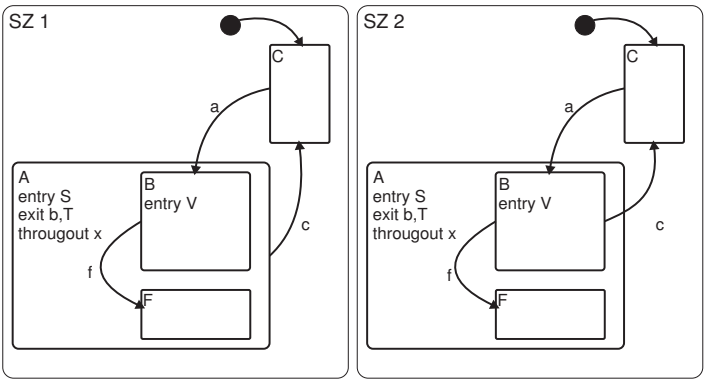

Abbildung 3.12.: Zustandsübergänge in einem SC, links beginnend an einem Superzustand, rechts in einem Superzustand

Innerhalb eines Superzustands können auch parallele Zustandsautomaten notiert sein. In Abbildung 3.11 sind zwei parallele SC als Komponenten dargestellt, die durch die gepunktete Linie getrennt sind. Jede Komponente arbeitet unabhängig voneinander. Falls eine Synchronisation notwendig ist, wird diese durch gemeinsame Zustandsübergänge erzwungen [72]. Alternativverzweigungen sind durch Kreuzungen oder Entscheidungen modelliert. Kreuzungen beschreiben statische und Entscheidungen dynamische Verzweigungen [93].

Abstraktions- und Zuordnungsmodell Die OMG bietet einen ganzheitlichen Ansatz zur modellgetriebenen Entwicklung, der auch als modellgetriebene Architektur bezeichnet wird. Die UML bzw. die SysML stellt mit ihren verschiedenen Meta-Modellen die Modellierungssprachen bereit. Dies bedeutet, dass eine Kombination der Meta-Modelle den Entwurfsprozess vom Engineering erster Ideen bis hin zu ausführbaren Modellen begleitet. Hierzu stehen eine Reihe von Modelltransformationen zur Verfügung. In der UML können zudem Profile definiert werden, die eine domänenspezifische Sprache festlegen. Eine durchgängige Verwendung der Objektorientierung in der UML ermöglicht die Verwendung von Konzepten wie Klasse und Instanz bzw. Vererbung. Die einzelnen Diagramme der UML sind nur unterschiedliche Sichten auf das gleiche Gesamtmodell [4]. SC im Speziellen finden in allen Planungsphasen Verwendung [80].

Aktions- und Aktivitätenmodell SC sind eine Erweiterung von EA und kombinieren die Ideen von Moore und Mealy (vgl. Kapitel 3.3.1, S. 28). Demnach ist es möglich Aktionen sowohl in einem Zustand als auch bei einem Zustandsübergang auszuführen. Unter einer

Aktion wird in diesem Kontext ein instantanes Ereignis, das idealisiert keine Zeit zur Ausführung benötigt, verstanden. Neben Aktionen können in Schritten auch Aktivitäten enthalten sein. Aktivitäten haben eine zeitliche Dauer. Ihr Start bzw. ihre Beendigung erfolgt durch Aktionen. Ein Attribut der Aktivität gibt an, ob die Aktivität zum aktuellen Zeitpunkt gestartet ist [72].

Ein Zustandsübergang kann aus fünf Teilen bestehen: dem Quellzustand, dem Ereignistrigger, der überwachten Bedingung, dem Effekt und dem Zielzustand. Eine Transition beginnt immer an einem Quellzustand und endet im Zielzustand. Der Ereignistrigger kennzeichnet das Ereignis, welches den Zustandsübergang einleitet. Trifft das Ereignis ein, wird die überwachte Bedingung, ein boolescher Ausdruck, ausgewertet. Die überwachte Bedingung ist demnach die Erlaubnis zum Feuern der Transition. Während des Zustandsübergangs können Aktionen, die sogenannten Effekte, ausgeführt werden [11].

Als Ereignis wird hier

> *„die Spezifikation eines signifikanten Vorkommens, das sich zeitlich und räumlich zuordnen lässt [, verstanden]. Im Kontext von Zustandsautomaten ist ein Ereignis ein Stimulus, der eine Zustandsänderung auslösen kann“ [11, S. 336].*

Ein solches Ereignis kann sowohl innerhalb eines SC als auch extern ausgelöst werden [11]. Es sind vier verschiedene Ereignisse in der UML definiert: Signale, Aufrufe, Abläufe von Zeitspannen und Zustandsänderungen. Des Weiteren finden generische Ereignisse Verwendung, die auf alle vier Ereignistypen reagieren. Signale ermöglichen die Modellierung eines asynchronen Nachrichtenaustauschs. Ein Aufruf kennzeichnet den Eingang einer Anfrage zur Ausführung einer Operation. Abläufe von Zeitspannen beschreiben ein zeitliches Ereignis. Ein Zustandsübergang entsteht, sobald ein boolescher Ausdruck von *falsch* nach *wahr* wechselt [93].

Ausführungssteuerungsmodell In einem SC darf in jeder Komponente nur ein Zustand aktiv sein. In jeder Komponente wird zunächst der Startpunkt getriggert, der den Startzustand aktiviert. Die Transition zwischen Startpunkt und Startzustand darf keinen Guard und kein Event besitzen. In diesem Zustand bleibt die Komponente, bis ein Zustandsübergang feuert. Hierzu muss ein Ereignis empfangen werden. Dabei gilt es zu beachten, dass die Bedingung eines Zustandsübergangs nur ausgewertet wird, wenn Guard true ist [93].

Der Übergang zu einem Superzustand kann zum Superzustand oder zu einem Unterzustand erfolgen. Im ersten Fall wird der Anfangs-Unterzustand des Superzustands aktiviert [72]. Des Weiteren ist es möglich, dass der letzte aktive Zustand wieder aktiviert wird, wenn ein Historian-Verbinder existiert [80]. Zur Vereinfachung können Verbindungspunkte für bedingte und selektive Eingänge verwendet werden [72]. Solange ein Unterzustand aktiv ist, ist auch der Superzustand aktiv [116].

Beginnt ein Zustandsübergang an einem Superzustand, kann jeder der Unterzustände des Superzustands über den Übergang verlassen werden. Beginnt er jedoch an einem Unterzustand, kann nur genau dieser Unterzustand verlassen werden. Bezogen auf Abbildung 3.12 bedeutet dies, dass der SC *SZ 1* im Zustand F durch ein Ereignis c in den Zustand C wechselt, der SC *SZ 2* weiter in F bleibt [72]. Wird ein Superzustand verlassen, werden auch alle enthaltenen Unterzustände deaktiviert [116].

Die Ausführung von Aktionen erfolgt sowohl beim Betreten als auch beim Verlassen eines Zustands. Dies wird durch die Worte *entry* bzw. *exit* gekennzeichnet. Die Steuerung einer

Aktivität x kann durch die Aktionen $start(x)$ und $stop(x)$ erfolgen. Soll eine Aktivität so lange laufen, wie der Zustand aktiv ist, lässt sich dies abkürzend mit dem Wort *throughout* (in der UML mit *do* bezeichnet) kennzeichnen [72]. Ereignisse können auch Aktionen triggern, ohne dass ein Zustandswechsel durchgeführt wird. Dies ist eine Verallgemeinerung von *entry, do* und *exit* [93].

Das Fehlen einer Übergangsbedingung für ein bestimmtes Ereignis führt nicht zu einem undefinierten Zustand wie bei den EA, sondern zu einem Verharren im derzeitigen Zustand, bis ein neues Ereignis eintritt [11]. Ereignisse können jedoch verzögert werden, d. h., sie werden bis zur Verwendung gespeichert [93].

3.3.8. PLC Statecharts

PLC Statecharts (PLC SC) erweitern die SC um eine Ausführungssemantik, die in einer SPS verwendet werden kann. Hierzu wird auf die Verwendung von Ereignissen verzichtet. Die Kommunikation läuft stattdessen über Signale ab. Die Vergabe von Prioritäten an die Transitionen erzielt ein deterministisches Ablaufverhalten [62]. Es wird zwischen mehrzyklischen und zyklusinternen PLC SC unterschieden. PLC SC sind formal zu analysieren und unter dem Namen PLC-Modeling Language (plcML) als UML-Profil definiert. In Abbildung 3.13 ist die Ähnlichkeit zu SC zu sehen (vgl. Abbildung 3.11). Die weiteren Erläuterungen in diesem Kapitel basieren auf [192].

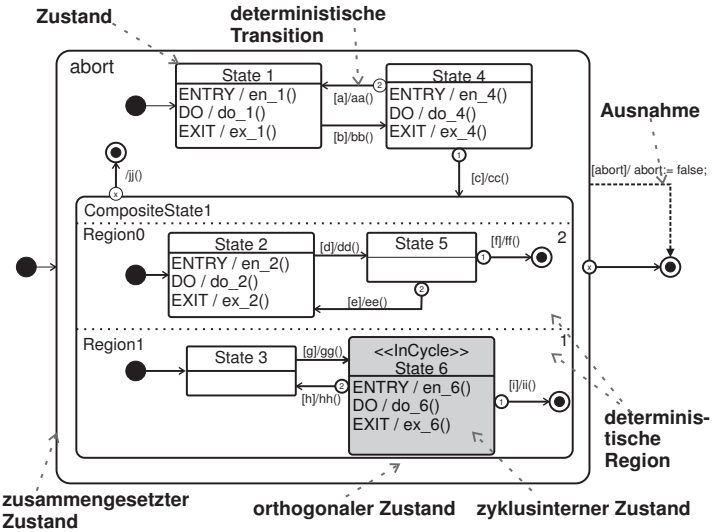

Abbildung 3.13.: Beispiel für ein PLC SC (nach [192])

Aufbaumodell PLC SC unterscheiden zwischen zyklusinternen und mehrzyklischen Zuständen. Mehrzyklische Zustände entsprechen den Zuständen der SC (weiß in Abbil-

dung 3.13 dargestellt). Zyklusinterne Zustände (grau in Abbildung 3.13 dargestellt) besitzen ein Attribut, das die Anzahl der DO-Aufrufe in einem Zyklus beschränkt. Deterministische Transitionen sind von den Transitionen aus den SC abgeleitet und erweitern diese um eine Prioritätsangabe. Initialzustand, Auswahlzustand, Gabelungszustand und Kreuzungsknoten werden als Pseudozustand modelliert.

Hierarchie- und Vernetzungsmodell Analog zu einem SC sind in einem PLC SC Zustände durch Übergänge, die sogenannten Transitionen, verbunden. Unterbrechungstransitionen, die an zusammengesetzte oder an orthogonale Zustände angebunden werden, ergänzen die PLC SC. Sie können nur dann an einfache Schritte angebunden sein, wenn diese eine DO-Aktivität haben, die unterbrochen werden soll.

Das Hierarchie- und Vernetzungsmodell wird gegenüber dem der SC wie folgt eingeschränkt: Zusammengesetzte Zustände besitzen keine Aktionen oder Aktivitäten. Hier erfolgt eine Zuordnung zu den in dem zusammengesetzten Zustand enthaltenen Zuständen. Des Weiteren ist exakt ein Initialzustand erforderlich, wenn ein zusammengesetzter Zustand durch eine Transition erreichbar ist. Verlassen werden muss ein zusammengesetzter Zustand über eine deterministische Transition, die durch ein Beendigungsereignis feuert.

Orthogonale Zustände besitzen ebenfalls keine Aktionen oder Aktivitäten. Sie dürfen nur deterministische Regionen beinhalten. Deterministische Regionen berücksichtigen die üblicherweise nicht-parallele Ausführung von SPS-Programmen und ermöglichen eine deterministische Serialisierung der parallelen Abläufe durch Angabe von Prioritäten. Ein orthogonaler Zustand wird über eine Transition verlassen. Hierzu müssen alle deterministischen Regionen einen Endzustand haben.

Eine weitere Rahmenbedingung der PLC SC besteht darin, dass zyklusinterne PLC SC nur aus zyklusinternen Zuständen bestehen dürfen. Das bedeutet, sobald ein PLC SC einen mehrzyklischen Zustand enthält, ist auch der PLC SC mehrzyklisch.

Abstraktions- und Zuordnungsmodell Große Teile des Abstraktions- und Zuordnungsmodells der plcML befassen sich mit Klassendiagrammen zur Modellierung der statischen Software-Strukturen und sind daher für diese Arbeit nicht relevant. Ablaufdiagramme können auf IEC 61131-3-kompatible Strukturen abgebildet werden. In Abbildung 3.14 ist dieser Ablauf der automatischen SPS-Codegenerierung basierend auf PLC SC dargestellt. PLC SC unterstützen zudem rollenbasierte Sichten auf das Modell. Dies hilft verschiedenen Nutzergruppen, nur die Bereiche eines PLC SC zu sehen, die für sie von Interesse sind. Tiefergehende Informationen über die Transformation sind auch [193] zu entnehmen.

Aktions- und Aktivitätenmodell Eine Aktion im PLC SC entspricht der Aktion im SFC, so dass hier eine Kompatibilität existiert. Aktivitäten haben einen booleschen Eingangsparameter zum Initialisieren und einen Ausgangsparameter, der anzeigt, dass die Aktivität abgelaufen ist. In der IEC 61131-3 sind keine Ereignisse vorgesehen. Daher sind in den PLC SC verschiedene Mechanismen enthalten, wie die Ereignisse in einer SPS emuliert werden können (vgl. Tabelle 3.3).

Ausführungssteuerungsmodell Das Ausführungssteuerungsmodell von PLC SC ist in [192] nicht enthalten, so dass die Darstellung in [194] als Grundlage für die Erläuterung an dieser Stelle gewählt wird. Die Ausführung ist in den SPS-Zyklus (vgl. Kapitel 2.2.3,

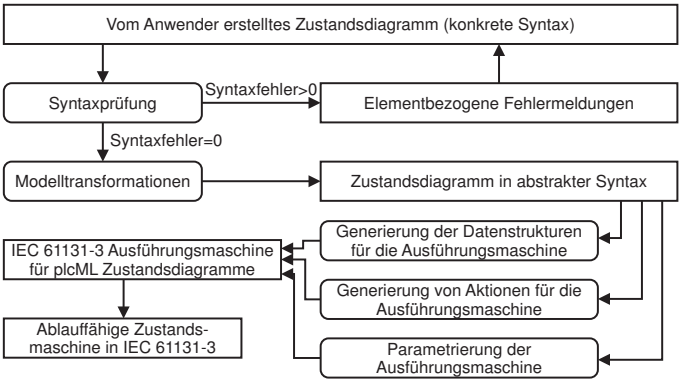

Abbildung 3.14.: Schema für die Codegenerierung aus PLC SC (nach [192])

S. 10) eingebunden. Die Ausführungslogik ist in der Sprache UPPAAL definiert und formal überprüft. Es findet eine Unterscheidung zwischen mehrzyklischen und zyklusinternen Zuständen statt. Wird ein mehrzyklischer Zustand betreten, werden zunächst die Aktionen im ENTRY-Bereich ausgeführt. Anschließend erfolgt in jedem Zyklus die einmalige Ausführung der DO-Aktivitäten und die Überprüfung der Transitionsbedingungen in absteigender Priorität. Feuert eine Transition, ist der Zyklus beendet. Unterbrechungstransitionen bewirken hingegen keinen Zykluswechsel. Zyklusinterne Zustände zeigen ein ähnliches Verhalten, der maximale Aufruf der DO-Aktivitäten ist jedoch beschränkt. Wird diese Anzahl überschritten, liegt ein Fehler vor.

3.3.9. Sequential State Charts

SSC sind eine Kombination aus SFC und SC, die die Vorteile der beiden Sprachen (z. B. die Lauffähigkeit in einer SPS von SFC und die Verwendung von Aktionsaufrufen in SC) kombiniert. Zur Vereinfachung der Sprache sind Konzepte wie z. B. nebenläufige Schrittketten oder Aktionsaufrufe in Transitionen nicht zugelassen. Des Weiteren enthalten SSC die Möglichkeit der dienstbasierten Kommunikation, die Modellierung der Prozeduren erfolgt in einem Ausführungsrahmen. Auch wenn SSC als Modell für allgemeine Prozeduren vorgestellt werden, ist ihre Einsatzfähigkeit auf die Steuerung technischer Prozesse beschränkt. SSC sind in [196, 197] eingeführt worden. Aus diesen Quellen stammt auch die folgende Zusammenfassung.

Aufbaumodell Ein SSC ist aus Schritten und Transitionen zusammengesetzt. Ein SSC sollte einen Startschritt besitzen und kann mehrere Endschritte haben. Alle Elemente eines SSC sind in einen Ausführungsrahmen integriert. Innerhalb des Ausführungsrahmens können auch weitere Funktionsbausteine verwendet werden.

Tabelle 3.3.: Emulierung von Ereignissen in einer SPS

Ereignis	Ersetzung
Change-Event	Es wird nicht der absolute Wert ausgewertet, sondern der Übergang zwischen zwei Werten. Hier kann beispielsweise der RTRIG-Baustein aus der VDI/VDE 3696 [180] Verwendung finden.
Completion-Event	Aktivitäten erhalten einen booleschen Ausgang, der *wahr* wird, wenn die Aktivität beendet ist. Dieser Wert kann in einer Transition abgefragt werden.
Time-Event	Da der Zeitpunkt t_0 der Aktivierung eines Schritts gespeichert ist, kann das Ereignis „nach $t = 10\,s$“ in einen Vergleich $t - t_0 > 10\,s$ umgewandelt werden. Zu beachten ist, dass es aufgrund der zyklischen Abarbeitung einer SPS nicht sinnvoll ist, einen Vergleich von Zeiten auf Gleichheit durchzuführen. Ein Beispiel ist die Abfrage $t = 0{,}5\,s$ bei einer Zykluszeit von $t_Z = 0{,}2\,s$.

Hierarchie- und Vernetzungsmodell Die Verbindung von Schritten und Transitionen erfolgt alternierend. Die Verbindung zweier Schritte über implizite Transitionen, die grafisch nicht dargestellt werden, ist jedoch auch möglich. SSC bieten die Möglichkeit, Unterprozeduren zu definieren. Eine solche Unterprozedur kann als eine eigene Prozedur mit einem eigenen Ausführungsrahmen definiert sein. Die Unterprozedur kann zudem in einem Ausführungsrahmen innerhalb des Ausführungsrahmens der Hauptprozedur enthalten sein. Eine Unterprozedur startet entweder durch Setzen der Variable `EN` oder durch einen Dienstaufruf (wenn die Prozedur im Ausführungsrahmen der Hauptprozedur enthalten ist).

Wird der aufrufende Schritt der Hauptprozedur verlassen, stoppt die Ausführung der Unterprozedur und diese bleibt im aktuellen Schritt stehen. Beim nächsten Aufruf startet die Unterprozedur an dieser Stelle. Die Hauptprozedur kann die Unterprozedur durch ein Kommando zurücksetzen. Falls die Unterprozedur nicht beendet werden darf, muss die ausgehende Transition des aufrufenden Schritts explizit den Endschritt der Unterprozedur abfragen.

Die Definition von Alternativverzweigungen ist möglich. In einer Alternativverzweigung können die ausgehenden Transitionen mit einer Priorität versehen sein. Sind bei der Ausführung der beschriebenen Prozedur mehrere Transitionsbedingungen erfüllt, feuert die Transition mit der höchsten Priorität. Ist keine Priorität angegeben, feuert die am weitesten links gezeichnete Transition.

Nebenläufige Prozeduren sind in SSC innerhalb eines Ausführungsrahmens nicht möglich. Die einzige Möglichkeit Funktionen nebenläufig auszuführen ist der zeitgleiche Aufruf mehrerer Unterketten. Die Synchronisation muss explizit in den Transitionsbedingungen modelliert sein.

Abstraktions- und Zuordnungsmodell Ein wichtiger Aspekt bei der Entwicklung der SSC stellt der einfache Entwurfsprozess dar. Hier ist ein Whitebox-Ansatz ausgewählt worden [199], d. h., jeder Ausführungsrahmen entspricht einem Funktionsbausteindiagramm

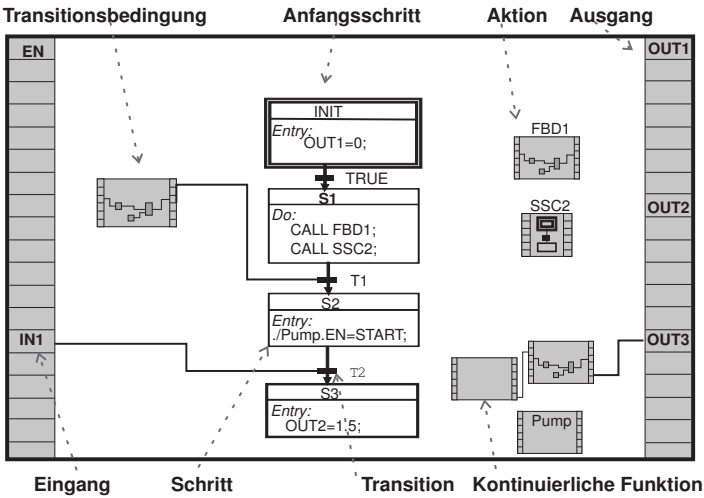

Abbildung 3.15.: Beispiel für einen SSC (nach [197])

nach der IEC 61131-3 [23]. Die interne Struktur des SSC kann sowohl im Engineering als auch zur Laufzeit erkundbar sein. Es ist jedoch auch möglich, den inneren Aufbau eines SSC zu verbergen. Auf diese Weise kann ein SSC als gekapseltes Modul verwendet werden.

Aktions- und Aktivitätenmodell Die Schritte eines SSC können Aktionen beinhalten. Eine Aktion kann hierbei

- das Setzen einer Ausgangsvariable des Ausführungsrahmens,
- das Setzen des Eingangs eines Funktionsbausteins innerhalb des eigenen Ausführungsrahmens oder
- der Aufruf eines lokalen Funktionsbausteins sein.

Es werden drei Ausführungszeitpunkte unterschieden. Zum einen ist eine einmalige Ausführung von Aktionen bei der Aktivierung (*entry*) oder beim Verlassen des Schritts (*exit*) möglich. Zum anderen können Aktionen zyklisch (*do*) ausgeführt werden, während der Schritt aktiv ist. Die Ausführung von Aktivitäten wird nicht direkt unterstützt.

Ausführungssteuerungsmodell Das Ausführungssteuerungsmodell der SSC ist im Ausführungsrahmen enthalten. Der Ausführungsrahmen kann als Funktionsbaustein verwendet werden. Er hat mindestens einen Eingang `EN` und zwei Ausgänge `ActualState` und `terminated`. Über den Eingang `EN` lässt sich der Ausführungsrahmen mit Hilfe verschiedener Kommandos der SSC in seinem Verhalten steuern. Der Ausgang `ActualState` gibt den momentan aktiven Schritt der Hauptkette nach außen weiter. Der Ausgang `terminated`

indiziert, dass die Prozedur in einen Endschritt gelaufen ist. In einem Ausführungsrahmen eines SSC können auch kontinuierliche Steuerungsfunktionen enthalten sein, die z. B. Berechnungen beinhalten können.

Wenn ein SSC aktiv ist, erfolgt in jedem Zyklus ein fester Ablauf. Zunächst werden die Eingänge gelesen und anschließend die interne Logik an den Eingängen ausgewertet. Nun liegt der Fokus auf dem aktiven Schritt. Wird der Schritt zum ersten Mal ausgeführt, werden alle Aktionen unter *entry* sequenziell ausgeführt. Anschließend erfolgt die Auswertung der ausgehenden Transitionen. Feuert eine Transition, werden die Aktionen unter *exit* sequenziell ausgeführt, der nächste Schritt aktiviert und dessen Aktionen unter *entry* ausgeführt. Feuert keine Transition, werden die Aktionen unter *do* ausgeführt. Unabhängig von der Prozedur führt die Ausführungssteuerung am Ende eines Zyklus die kontinuierlichen Steuerungsfunktionen aus.

3.4. Prozedurbeschreibungssprachen zur Steuerung von Geschäftsprozessen

Während die Steuerung technischer Prozesse essentiell für den Erfolg der Prozesse ist, sind Geschäftsprozesse lange Zeit implizit gesteuert worden. Erst seit Anfang der 1990er Jahre erfolgt eine systemische Aufzeichnung und Optimierung von Geschäftsprozessen [102]. Die Dokumentation der Geschäftsprozesse ist Basis für viele Zertifizierungen eines Unternehmens bzw. seiner Mitarbeiter. Hier sind z. B. das OMG Certified Expert in Business Process Management (OCEB)-Programm [190] oder die Zertifizierung nach ISO 9001 [33] zu nennen. Ursachen für den immer stärkeren Zwang zum Geschäftsprozessmanagement liegen in den immer komplexer werdenden Abläufen, die unternehmensübergreifend sind und in die fortlaufend neue Informationstechnik integriert werden [173]. Es wird zwischen der fachlichen Modellierung und der technischen Modellierung von Geschäftsprozessen unterschieden. Bei der fachlichen Modellierung ist das erzeugte Modell nur mit Hilfe des Menschen interpretierbar, da die Modelle semiformal, natürlichsprachig oder unvollständig sind. Bei der technischen Modellierung entstehen durch Computer ausführbare Modelle [101].

Im Folgenden wird der Begriff „Prozedur“ verwendet, wenn der Steuerungsaspekt eines Geschäftsprozess gemeint ist. Als Modellierungssprachen werden Aktivitätsdiagramme (AD), BPEL, Business Process Modeling Notation (BPMN), Koordination, Kooperation und Kommunikation (K3) und Ereignisgesteuerte Prozessketten (EPK) in dieser Reihenfolge vorgestellt. In der IEC 62264[11] wird keine Modellierungssprache definiert. Vielmehr werden die Datenflüsse innerhalb und zwischen den Systemen der Ebenen 3 und 4 in den Teilen 1 und 3 [27, 29] sowie die Inhalte des Austauschs in den Teilen 2 und 4 [28, 30] spezifiziert.

3.4.1. Aktivitätsdiagramme

AD sind einer der Diagrammtypen in der UML, die es erlauben das Verhalten eines dynamischen Systems zu beschreiben [11]. AD basieren auf Flussdiagrammen aus den 1960er

[11] Die IEC 62264 basiert auf der ISA 95.

Jahren [4] und sind semantisch ähnlich zu Interaktionsdiagrammen. Während Interaktionsdiagramme auf den Austausch von Nachrichten zwischen Objekten spezialisiert sind, liegt der Blickwinkel bei den AD auf der Modellierung der sequentiellen und nebenläufigen Schritte in der Ablaufbeschreibung [11]. Das hier vorgestellte Meta-Modell bezieht sich auf die UML 2-Version, in der das Konzept vollständig überarbeitet worden ist [93]. Im SysML-Profil werden AD im Vergleich zur UML Kontrolloperatoren hinzugefügt, die ein Eingreifen in den Ablauf von außen ohne eingehenden Kontrollfluss ermöglichen [4]. In Abbildung 3.16 ist ein Beispiel für ein AD dargestellt.

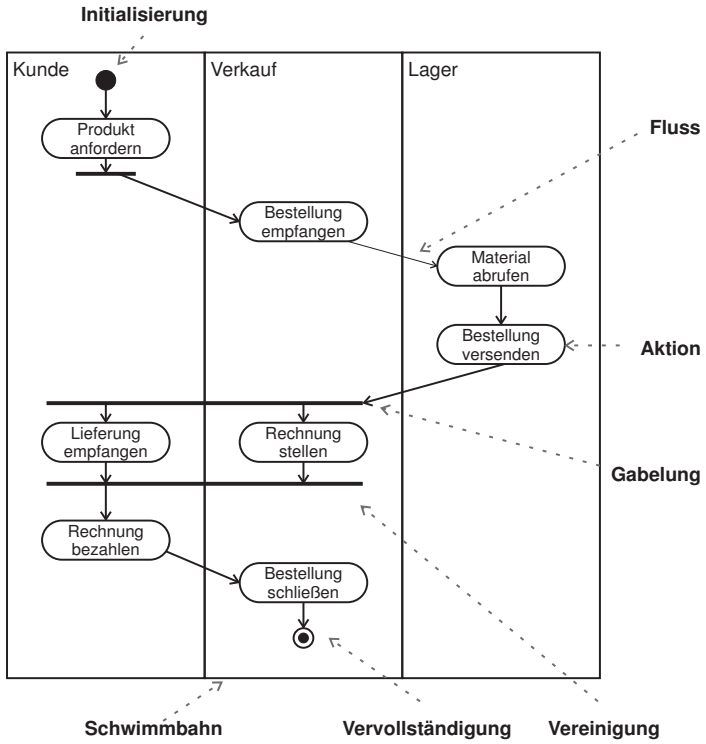

Abbildung 3.16.: Beispiel für ein AD (nach [11])

Aufbaumodell AD bestehen aus Aktionen, Aktivitätsknoten, Flüssen und Objektwerten [11]. Aktionen sind atomare Elemente, die sich nicht weiter zerlegen lassen bzw. deren Zerlegung für das Modell uninteressant ist [93]. Aktivitätsknoten lassen sich weiter in ausführbare Knoten, Objektknoten und Kontrollknoten unterteilen. Zu den ausführbaren Knoten zählen die Elemente mit ausführbaren Tätigkeiten. Objektknoten dienen als Datenspeicher. Kontrollknoten strukturieren und steuern den Ablauf des AD. Flüsse können

Kontrollflüsse oder Objektflüsse sein [93]. Des Weiteren sind in einem AD Initialisierungen (Startknoten) und Vervollständigungen (Endknoten) enthalten. Jedes AD muss mindestens eine Initialisierung beinhalten. Vervollständigungen sind optional [11]. Es wird zwischen Flussende und Aktivitätsende unterschieden. Ein Flussende terminiert ausschließlich den betreffenden Fluss, während ein Aktivitätsende alle Kontrollflüsse beendet [93].

Hierarchie- und Vernetzungsmodell Die Aktionen sind über Flüsse miteinander verbunden. Aktivitätsknoten stellen Gruppierungen von Aktionen oder weiteren Aktivitätsknoten dar und ermöglichen daher den Aufbau einer hierarchischen Struktur. Die Aktionen und Aktivitäten können in Schwimmbahnen enthalten sein (vgl. Abbildung 3.16). Schwimmbahnen dienen ausschließlich der Übersichtlichkeit und haben keine weitere Funktion als die optische Gliederung [11].

In einem AD können alternative und nebenläufige Pfade definiert werden. Verzweigungen, die einen eingehenden und mindestens zwei ausgehende Flüsse haben, öffnen alternative Pfade. An den ausgehenden Flüssen sind boolesche Variablen notiert, die vollständig und überdeckungsfrei sein müssen. Am Ende der alternativen Pfade folgt eine Zusammenführung. Gabelungen eröffnen nebenläufige Pfade und Vereinigungen führen sie wieder zusammen. Dies schließt echte Nebenläufigkeit und verzahnte sequentielle Abläufe ein [11].

Abstraktions- und Zuordnungsmodell AD lassen sich in allen Phasen der Softwareentwicklung nutzen [93]. Nach [11] werden im Entwurfsprozess von AD zunächst alle Schwimmbahnen auf einer geeigneten Ebene definiert. In diese Schwimmbahnen platziert der Entwickler die Aktionen, die er über Flüsse miteinander verbindet. Hierbei sollen zunächst sequentielle Abläufe, anschließend alternative Pfade und zuletzt die Nebenläufigkeiten erzeugt werden. Häufig verwendete Gruppen von Aktionen sollen zu Aktivitäten kombiniert werden. Es bietet sich an Aktivitäten in einer Bibliothek abzulegen, so dass diese Aktivitäten wiederverwendbar sind [4]. Eine automatische Codegenerierung ist möglich [11].

Aktions- und Aktivitätenmodell Es ist keine einheitliche Syntax für Aktionen festgelegt. Aktionen können jedoch fest mit anderen Objekten im selben UML-Modell verknüpft sein. Flüsse haben keine Übergangsbedingung [11]. Sofern es sich um Objektflüsse handelt, kann das Objekt mit seinem Zustand beschrieben werden. Aktionen können Vor- und Nachbedingungen haben, die einen Zustand vor bzw. nach der Ausführung der Aktion beschreiben [93].

Ausführungssteuerungsmodell Das Ausführungssteuerungsmodell der AD basiert ähnlich wie das der PN auf Token. Diese sind in der grafischen Darstellung allerdings nicht sichtbar. Startknoten erzeugen jeweils ein Token, das dem jeweiligen Kontrollfluss folgt. Erreicht es eine Aktion, wird diese ausgeführt [4]. Die Aktionen in einem AD haben idealisiert keine zeitliche Dauer. Das Token aktiviert das nächste Element im Fluss, sobald die vorherige Aktion oder Aktivität ausgeführt worden ist [11]. Gabelungen vermehren und Vereinigungen verringern die Anzahl der Token. Das Token wandert solange, bis es auf einen Endknoten trifft [4]. Dieser Ablauf kann durch Abbrüche beeinflusst werden. Hierzu ist die Definition von Unterbrechungsbereichen notwendig [93].

3.4.2. Business Process Execution Language

Die BPEL ist eine Modellierungssprache zur Beschreibung und Ausführung von Prozeduren zur Steuerung von Geschäftsprozessen [101]. Sie basiert auf der XML und auf Web-Diensten [132]. Hauptziele bei der Entwicklung der BPEL sind die Flexibilität und die Wartbarkeit der erzeugten Modelle gewesen [188]. In der Version 1.0, auch als BPEL for Web Services (BPEL4WS) bezeichnet, hat die Modellierung vollständig automatisierter Prozeduren im Vordergrund gestanden. Erst in der zweiten Version (durch die Organization for the Advancement of Structured Information Standards (OASIS) als Web Service BPEL (WS-BPEL) standardisiert) ist die Interaktion mit menschlichen Akteuren integriert worden [101]. WS-BPEL nutzt verschiedene Spezifikationen aus der XML. Die Web Service Description Language (WSDL)-Nachrichten und die XML Schema Definition (XSD) stellen das Datenmodell bereit, während XPath und Extensible Stylesheet Language Transformation (XSLT) Datenmanipulationen unterstützen [132].

Aufbaumodell Der Aufbau eines BPEL-Modells ist im Prozesskompositionsmodell beschrieben [188]. Ein BPEL-Modell besteht demnach aus einer Prozedur (im BPEL als Prozess bezeichnet), der den XML-Namensraum (vgl. Kapitel 4.3.2, S. 84) festlegt. Die Prozedurbeschreibung enthält zunächst die in WSDL formulierten Schnittstellen. Die Schnittstellenbeschreibung umfasst sowohl die Schnittstelle zum Modell selber als auch die Schnittstellen zu den aufzurufenden Diensten. Die Verknüpfung zwischen BPEL- und WSDL-Datei erfolgt durch sogenannte Partner-Links. Globale Variablen können ebenfalls in der Prozedurbeschreibung definiert sein [101]. Die eigentliche Prozedur wird mit Hilfe von Aktivitäten modelliert, wobei zwei Klassen differenziert werden, Basisaktivitäten und strukturierte Aktivitäten. Basisaktivitäten beschreiben die elementaren Steuerungsschritte, die nicht weiter zerlegbar sind [132].

Hierarchie- und Vernetzungsmodell In der BPEL sind nur Kontrollflüsse explizit modelliert, der Datenfluss erfolgt über die globalen Variablen. Sequenzen sind die einfachste Vernetzungsart in der BPEL. Die Anordnung der Aktivitäten in der XML-Datei gibt die Reihenfolge der Ausführung vor. Alternativverzweigungen sind durch Wenn-Dann-Blöcke modelliert. Flussaktivitäten kapseln parallel auszuführende Objekte. Des Weiteren sind Schleifen wie in anderen Programmiersprachen möglich [101].

Abstraktions- und Zuordnungsmodell Mit der Hilfe von der BPEL lassen sich abstrakte und ausführbare Prozeduren modellieren [132]. Beide Typen werden mit Geschäftsprozess-Modellierungswerkzeugen entworfen. In abstrakten Prozeduren sind wesentliche Details zur Ausführung weggelassen. Sie dienen als Ausgangspunkt für ausführbare Prozeduren, aber auch als Dokumentation oder zum Know-How-Schutz [101]. Ausführbare Prozeduren können in der BPEL-Engine ausgeführt werden.

BPEL basiert auf einer typbasierten Entwicklung der Modelle. Die Dienste werden zunächst auf Typebene implementiert und können anschließend auch dort orchestriert werden [132].

Aktions- und Aktivitätenmodell Die Interaktion mit der Umgebung erfolgt nachrichtenbasiert. Aufrufobjekte starten die Dienste. Empfangsobjekte und Antwortobjekte empfangen die entsprechenden Antworten und reagieren auf diese. Die Anbindung zwischen

Prozedurmodell und Diensten kann sowohl statisch als auch dynamisch sein. Statische Verknüpfungen können bei der Modellierung oder beim Deployment angelegt werden. Für eine dynamische Allokation stehen Repositorys oder externe Datenquellen zur Verfügung [101].

Innerhalb einer BPEL-Prozedur werden Variablen durch Manipulationsobjekte verändert. Als Variablentypen können WSDL-Nachrichten, einfache bzw. komplexe XSD-Typen oder XSD-Elemente verwendet werden. In Sequenzen sind keine Übergangsbedingungen notwendig. Bei Alternativverzweigungen werden die Bedingungen standardmäßig in XPath formuliert [101].

Ausführungssteuerungsmodell Bei der Ausführung eines BPEL-Modell in einer BPEL-Engine wird eine neue Instanz des Modells erzeugt (vgl. Abbildung 3.17).

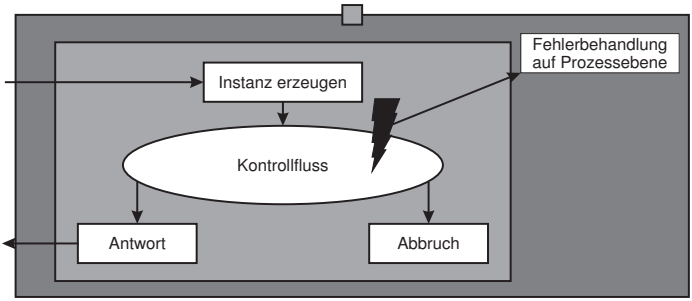

Abbildung 3.17.: Lebenszyklus einer BPEL-Instanz (nach [188])

Anschließend wird der Kontrollfluss gestartet und, falls kein Fehler auftritt, die Antwort gesendet [188]. Innerhalb der Ausführung ist sowohl ein synchrones als auch ein asynchrones Verhalten modellierbar. Je nach verwendetem Empfangsobjekt hält dieses den Kontrollfluss an, bis eine Antwort eingeht oder wartet im Hintergrund auf die eingehende Antwort, während schon die nächsten Dienste aufgerufen werden. Ist eine Flussaktivität im Kontrollfluss vorhanden, schaltet diese erst zum nächsten Objekt, wenn alle Pfade abgearbeitet sind [101].

Es wird zwischen fachlichen und technischen Fehlern unterschieden. Ein fachlicher Fehler ist beispielsweise eine falsche Eingabe, die das WSDL-Dokument behandeln muss. Ein technischer Fehler ist z. B. ein Timeout und muss in der BPEL-Spezifikation abgefangen werden [101]. Auf technische Fehler kann sowohl durch explizite Abbrüche im Kontrollfluss als auch durch einen Fehlerbehandlungsmechanismus auf Prozedurebene reagiert werden (vgl. Abbildung 3.17) [188].

3.4.3. Business Process Model and Notation

Die BPMN ist eine meta-modellbasierte Sprache zur formalen Beschreibung von Geschäftsprozessen. Die aktuelle Version 2.0 wird von der OMG gepflegt und ist in der ISO/IEC 19510 [83] standardisiert [101]. In der Definition sind nicht nur die Notation,

sondern auch die grafische Syntax und genaue Verknüpfungsregeln festlegt. Somit ist die BPMN eine grafische Modellierungssprache [60]. Die BPMN bietet somit eine standardisierte Möglichkeit der Beschreibung von Prozeduren zur Steuerung von Geschäftsprozessen. Ein zweiter Vorteil ist die direkte Ausführbarkeit der Modelle, die seit der Version 2.0 gegeben ist [66]. Die BPMN kann zum Entwurf neuer Prozeduren, als Ausgangspunkt für die Prozessverbesserung und für die Dokumentation bestehender Prozeduren genutzt werden [83]. In Abbildung 3.18 ist ein Beispiel für ein BPMN-Modell gezeigt.

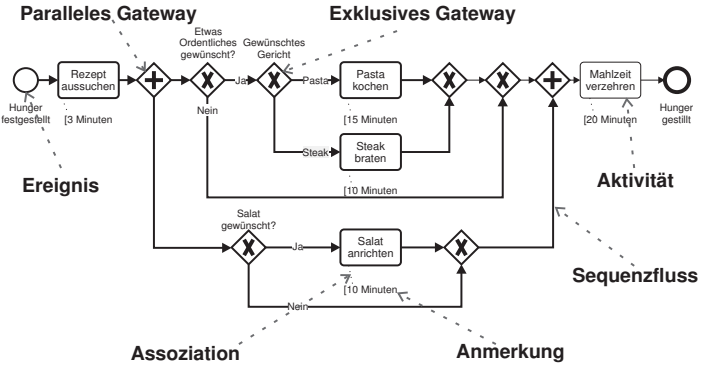

Abbildung 3.18.: Beispiel für ein BPMN-Modell (nach [66])

Aufbaumodell In der ISO/IEC 19510 sind eine Vielzahl verschiedener Elemente aufgeführt. Zur Komplexitätsreduktion wird eine Einteilung in Basiselemente und erweiterte Konzepte vorgenommen [195]. Die Basiselemente decken bereits eine Vielzahl von zu modellierenden Prozeduren ab, während die erweiterten Konzepte für Spezialfälle definiert sind [60]. Untersuchungen zur Verwendung der Elemente zeigen, dass in den allermeisten Fällen nur die Basisobjekte verwendet werden [201], so dass sich die weitere Beschreibung auf die Basiselemente beschränkt.

Die Basisobjekte lassen sich in fünf Kategorien[12] einteilen, Flussobjekte, Verbindungsobjekte, Daten, Schwimmbahnen und Artefakte [83]. Zu den Flussobjekten zählen Aktivitäten, Ereignisse und Gateways. Ereignisse lassen sich in Startereignis, Zwischenereignis und Endereignis spezifizieren. Als Verbindungselemente werden Sequenzflüsse, Nachrichtenflüsse und Assoziationen verwendet. Datenobjekte beschreiben die Daten, die in den Aktivitäten benötigt werden. Schwimmbahnen werden in Pools und Bahnen unterteilt. Artefakte sind Gruppen und Anmerkungen [60].

Hierarchie- und Vernetzungsmodell Die drei Verbindungsobjekttypen des Aufbaumodells stellen die Verbindung zwischen den Flussobjekten her. Der Sequenzfluss ist hierbei

[12] In [83, 195] stellen Daten eine eigene Kategorie dar, in [60] werden sie zu den Artefakten gezählt.

die Basis für die zeitliche Abfolge der Aktivitäten. Nachrichtenflüsse definieren einen Kommunikationskanal für verschiedene Prozessbeteiligte. Artefakte werden über Assoziationen an die Flussobjekte angebunden [60].

Neben dieser Möglichkeit zur Definition linearer Abläufe können mit Hilfe von Gateways komplexere Abläufe modelliert werden. Gateways werden sowohl für die Aufteilung als auch für die Zusammenführung verschiedener Ablaufpfade verwendet [83]. In den Basisobjekten sind drei Gatewaytypen enthalten, datenbasierte exklusive Gateways, parallele Gateways und datenbasierte inklusive Gateways. Datenbasierte exklusive Gateways beschreiben Entscheidungen, von denen jeweils genau eine getroffen werden muss. Parallele Gateways eröffnen bzw. schließen nebenläufige Pfade. Datenbasierte inklusive Gateways beschreiben Entscheidungen, von denen jeweils mindestens eine getroffen werden muss [66].

Hierarchien können durch Teilprozesse aufgebaut werden, Schleifen ermöglichen eine einfache Notation wiederholender Aktivitäten [101]. Ein Teilprozess stellt hierbei einen eigenen, vollständigen Ablauf dar. Eine organisatorische Aufteilung kann mit Pools und Lanes durchgeführt werden. Lanes können verschachtelt sein. So kann z. B. eine Abteilung eine Lane darstellen und die Mitarbeiter der Abteilung jeweils eine eigene Unter-Lane. Eine Aktivität muss genau einer Lane zugeordnet sein, wenn Lanes definiert sind. Ein Pool steht für eine übergeordnete Instanz, die die Steuerung des Ablaufs koordiniert [66].

Abstraktions- und Zuordnungsmodell Die BPMN ermöglicht die Modellierung abstrakter und ausführbarer Prozeduren [83]. BPMN unterstützt die Definition von globalen Teilprozeduren. Sie können durch die Oberprozedur mehrfach aufgerufen werden. Globale Teilprozeduren unterscheiden sich durch ihre Wiederverwendbarkeit von gewöhnlichen Teilprozeduren zur Erstellung einer hierarchischen Struktur. Während gewöhnliche Teilprozeduren automatisch Zugriff auf die Datenobjekte des Oberprozesses haben, muss die Zuordnung der Daten bei globalen Teilprozeduren explizit erfolgen [66].

Aktions- und Aktivitätenmodell Aktivitäten beschreiben Vorgänge, in denen Arbeit verrichtet wird [101]. Die Ausführung erfolgt durch Automaten oder durch Menschen. Jede Aktivität besitzt einen Zustand, der durch die Ausführungssteuerung verwaltet wird. Des Weiteren enthält ein Attribut die Anzahl der Token, die zur Ausführung der Aktivität benötigt werden [83].

Zwischenereignisse dienen dem Anhalten des Sequenzflusses an bestimmten Stellen. Ein Weiterschalten ist erst dann möglich, wenn das Zwischenereignis eingetreten ist. Zudem können Abbruchereignisse an Aktivitäten angefügt sein. Ein Eintreten des Ereignisses, während die Aktivität aktiv ist, bricht die Aktivität ab. Auch explizite Fehlerereignisse werden an Aktivitäten angefügt [66].

Ausführungssteuerungsmodell Die BPMN-Engine startet eine BPMN-Prozedur, sobald eines ihrer Startereignisse eintritt. Dazu wird eine Instanz der Prozedur angelegt. Die Ausführung der Aktivitäten wird durch Token gesteuert. Die Ausführung einer Aktivität erfolgt, sobald genügend Token anliegen. Nach der Ausführung schalten die Token weiter und die Aktivität wird deaktiviert [83]. Bei einem datenbasierten exklusiven Gateway schaltet das Token auf den Pfad, dessen Bedingung erfüllt ist. Der Ersteller des Modells muss darauf achten, dass die Bedingungen sich gegenseitig ausschließen. Eine Verklemmung

lässt sich durch die Auszeichnung eines Pfads als Standard-Pfad vermeiden. Parallele oder datenbasiert inklusive Gateways können Token erzeugen oder vernichten [66]. Eine Prozedurausführung ist beendet, wenn es keine Token mehr gibt und keine Aktivität mehr aktiv ist [83].

3.4.4. Koordination, Kooperation und Kommunikation

K3 ist eine Modellierungssprache, die Ende der 1990er Jahre als Erweiterung der AD entwickelt worden ist [95]. Der Fokus der Entwicklung hat auf der Modellierung von schwach strukturierten Abläufen gelegen. In Abbildung 3.19 sind die wichtigsten Notationselemente von K3 dargestellt.

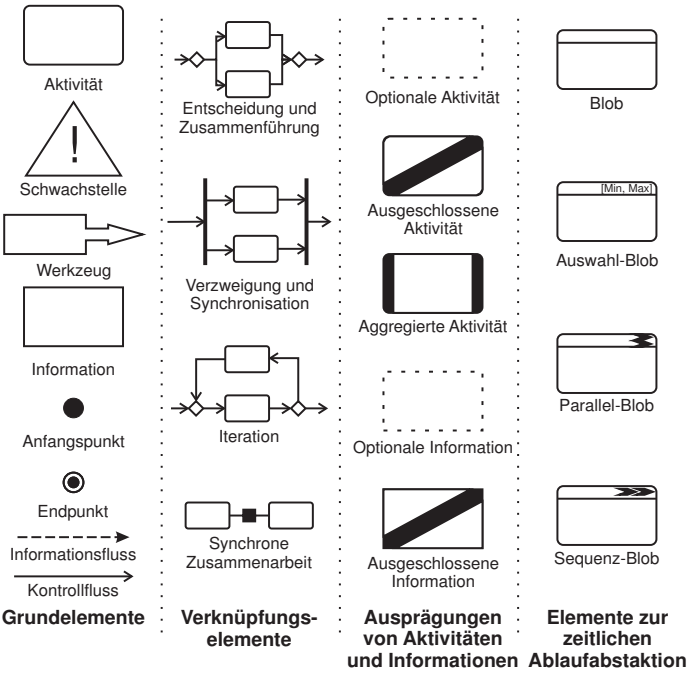

Abbildung 3.19.: Elemente eines K3-Graphen (nach [131])

Aufbaumodell K3 basiert auf den AD und verwendet daher die dort definierten Elemente Aktivität, Anfangs- und Endpunkt (vgl. Kapitel 3.4.1, S. 54) [95]. Diese Elemente können durch die sogenannten Satellitenelemente Information, Schwachstelle und Werkzeug weiter spezifiziert werden [131].

Hierarchie- und Vernetzungsmodell Ausgehend vom Anfangspunkt sind die Aktivitäten über Kontrollflüsse bis zum Endpunkt verbunden. Eventuell vorhandene Satellitenelemente sind über Informationsflüsse mit den zugehörigen Aktivitäten verbunden. Der Kontrollfluss lässt sich in mehrere Alternativen aufteilen, wobei die Iteration ein Spezialfall der Entscheidung ist [95]. Schwimmbahnen teilen das Modell in verschiedene Organisationseinheiten auf [65]. Innerhalb einer Organisationseinheit sind nebenläufige Prozeduren durch Verzweigungen und Synchronisationen modelliert, eine organisationsübergreifende Darstellung nebenläufiger Vorgänge wird über die synchrone Zusammenarbeit ermöglicht [95].

Aggregierte Aktivitäten stellen eine Möglichkeit der Einführung einer Hierarchie in K3 dar. Eine solche aggregierte Aktivität fasst als Makroschritt mehrere Aktivitäten zusammen [131]. Binary Large Objects (Blobs) unterstützen die Möglichkeit, schwach strukturierte Abläufe zu modellieren. In einem konventionellem Blob sind Aktivitäten enthalten, deren Ausführungsreihenfolge nicht ersichtlich ist. Ein Auswahl-Blob gibt an, wie viele Aktivitäten mindestens ausgeführt werden müssen und wie viele maximal ausgeführt werden dürfen. In einem Sequenz-Blob müssen alle Aktivitäten in beliebiger Reihenfolge nacheinander ausgeführt werden, in einem Parallel-Blob darf die Ausführung zeitgleich stattfinden [131].

Abstraktions- und Zuordnungsmodell Im Entwurfsprozess mit K3 werden zwei Dimensionen des Entwicklungsstands berücksichtigt, die Generalitätsebene und die Detailebene. Je genereller ein entworfenes Modell ist, desto mehr Systeme kann es beschreiben. In einem detaillierteren Modell sind detailliertere Informationen enthalten. Beispielsweise sind mehr Attributen Werte zugewiesen oder Aktivitäten sind durch aggregierte Aktivitäten bzw. Blobs ersetzt worden [45]

Aktions- und Aktivitätenmodell Ähnlich wie bei den EPK erfolgt die Interaktion mit der Umgebung über Informationsobjekte. Diese werden durch die Aktivitäten benötigt, bearbeitet oder erzeugt [65].

Ausführungssteuerungsmodell Die Ausführung eines K3-Modells erfolgt analog zu den AD. Ausgenommen hiervon ist die Ausführung von Blobs. Hier hat der Entwickler die Ausführungsreihenfolge im Prozedurdesign nicht festgelegt. Stattdessen soll der Ausführer der Prozedur diese zur Laufzeit bestimmen. Zudem können bei der Ausführung eines K3-Modells optionale und ausgeschlossene Aktivitäten auftreten. Optionale Aktivitäten können situationsbedingt ausgeführt werden, während die Ausführung ausgeschlossener Aktivitäten verboten ist [131].

3.4.5. Ereignisgesteuerte Prozessketten

EPK sind ein prozessorientierter Ansatz zur grafischen Beschreibung und Modellierung von Geschäftsprozessen. Sie sind hauptsächlich im deutschsprachigen Raum verbreitet [131]. Sie stellen einen kontrollflussorientierten Ansatz dar, der die Mängel der bestehenden datenflussorientierten Ansätze beheben soll. Der Fokus ist dabei sowohl auf die Analyse von Prozedurketten als auch auf die Modellierung der Beziehungen zwischen Datenobjekten

gelegt worden [94]. EPK werden insbesondere in der Architektur integrierter Informationssysteme (ARIS) zur Geschäftsprozessmodellierung verwendet [160]. In Abbildung 3.20 wird ein Beispiel für eine EPK dargestellt.

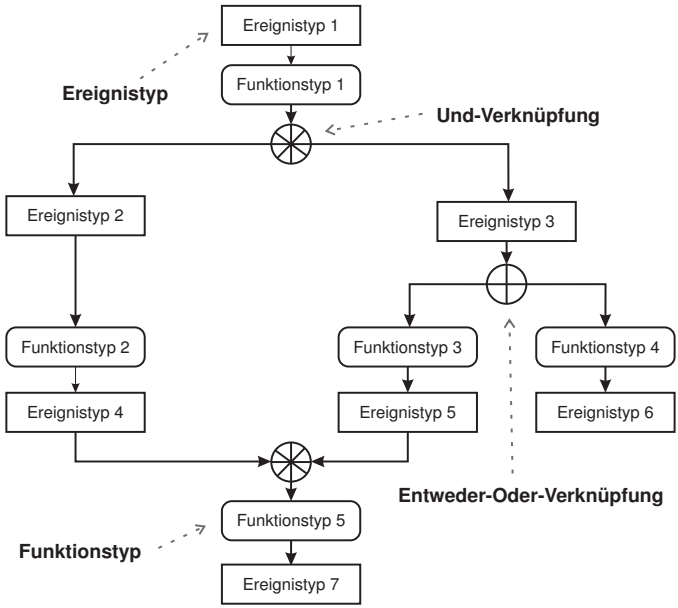

Abbildung 3.20.: Beispiel für eine EPK (nach [94])

Aufbaumodell Eine EPK besteht aus drei Konstrukten: dem Informationsobjekt, der Funktion und dem Ereignis. Informationsobjekte sind semantisch beschriebene Sachverhalte, die Mengen realer oder abstrakter Daten beschreiben. Funktionen sind aktive Komponenten, die eine Beschreibung eines Vorgangs darstellen. Sie sind semantische Regeln, die die Überführung von Eingangszuständen in definierte Ausgangszustände spezifizieren. Ereignisse sind passive Komponenten, die den Ablauf im Informationssystem beeinflussen, wenn ein definierter Zustand eingetreten ist [94]. Funktionen sind zeitverbrauchende Vorgänge, während sich Ereignisse auf einen konkreten Zeitpunkt beziehen [160].

Hierarchie- und Vernetzungsmodell In einer EPK lösen Ereignistypen die Funktionstypen aus. Funktionstypen erzeugen während ihrer Ausführung neue Ereignistypen [94]. Daher sind Funktions- und Ereignistypen alternierend verbunden [131], wobei jede EPK mit einem Ereignis beginnt und endet [160]. Des Weiteren unterstützen EPK drei verschiedene Verknüpfungsoperatoren: konjunktive, disjunktive und adjunktive Verknüpfungen (vgl. Abbildung 3.20).

- Bei einer konjunktiven Verknüpfung zweier Aussagen müssen beide Aussagen *wahr* sein, damit die verknüpfte Aussage *wahr* ist.
- Bei einer disjunktiven Verknüpfung zweier Aussagen muss genau eine Aussage *wahr* sein, damit die verknüpfte Aussage *wahr* ist.
- Bei einer adjunktiven Verknüpfung zweier Aussagen muss mindestens eine Aussage *wahr* sein, damit die verknüpfte Aussage *wahr* ist.

Die Verknüpfungen können sich auf Ereignistypen oder auf Funktionstypen beziehen [94]. Nach einem Ereignistyp darf jedoch keine disjunktive oder adjunktive Verknüpfung verwendet werden [160].

Hierarchische Strukturen können durch strukturbildende Objekte erzeugt werden. Zumeist werden Hierarchien jedoch durch das überlagerte ARIS-System erzeugt und die einzelnen EPK haben keine Hierarchie [160].

Abstraktions- und Zuordnungsmodell In einer EPK sind zwei Ebenen vorgesehen: die Abstraktionsebene und die Ausprägungsebene. In der Aktionsebene werden Ereignis- und Funktionstypen definiert, die eine Sammlung verschiedener Ausprägungen von Ereignissen und Funktionen darstellen [94].

Aktions- und Aktivitätenmodell Die Interaktion mir der Umgebung erfolgt über die Informationsobjekte. Die Funktions- und Ereignistypen sind über eine $n : m$-Beziehung mit Informationsobjekten verbunden [94].

Ausführungssteuerungsmodell Die EPK enthält keine eigenständige Beschreibung ihrer Ausführung. Dies erfolgt z. B. in Systemen wie ARIS [160]. Solche Systeme müssen bei der Ausführung einer EPK Abhängigkeiten zwischen der EPK, den Datenmodellen und den Funktionsstrukturen berücksichtigen. Funktionen können Daten anfordern, ihnen können Daten durch Nachrichten geschickt werden oder Ereignisse stellen selbst Informationsobjekte dar [94].

3.5. Vergleich der analysierten Sprachen

Die Analyse der Sprachen in den Kapiteln 3.3 und 3.4 zeigt viele Gemeinsamkeiten zwischen den Prozedurbeschreibungssprachen, stellt aber auch Unterschiede dar. Die Ergebnisse sind in den Tabellen in Anhang A, S. 105, zusammengefasst. Im Folgenden werden die analysierten Sprachen bezogen auf die Elemente einer Prozedurbeschreibungssprache nach Kapitel 3.1, S. 24, verglichen.

Aufbaumodell

Alle Sprachen bestehen aus zwei Basiselementtypen. Der erste Basiselementtyp beschreibt die Beharrungspunkte während eines Ablaufs und wird als Schritt, Zustand, Knoten oder Stelle bezeichnet. Der zweite Basiselementtyp, Transition oder Übergangsbedingung genannt, steuert den Sequenzfluss und damit die Aktivierung der Beharrungspunkte. Des Weiteren sind Anfangs- und Endpunkte spezielle Beharrungspunkte einer Prozedur.

Ein wichtiges Kriterium bei der Definition des Aufbaumodells ist eine möglichst geringe Zahl von Basiselementtypen. Beispielsweise ist bei der BPMN die Anzahl der verschiedenen Elementtypen zu hoch, so dass die Komplexität problematisch ist [131].

Hierarchie- und Vernetzungsmodell

Zur Erzeugung einer konkreten Prozedur werden in allen Sprachen Instanzen der Basiselementtypen festgelegt. Die Instanzen werden über Kanten miteinander verschaltet, so dass eine Ablaufbeschreibung vorliegt. In den meisten Sprachen sind nur Kanten von einem Basiselementtyp zum jeweils anderen erlaubt.

Mithilfe der Prozedurbeschreibungssprachen lassen sich nicht nur lineare Abläufe darstellen. Alternativverzweigungen ermöglichen die Modellierung von Entscheidungssituationen. Nebenläufige Pfade bzw. orthogonale Automaten definieren zeitgleich ablaufende Prozeduren. Die technische Umsetzung von Nebenläufigkeit ist jedoch problematisch. Sowohl Menschen als auch maschinelle Steuerungen, die typischerweise einen Prozessorkern besitzen, können nur eine Sache zur selben Zeit steuern. Daher müssen die nebenläufigen Prozeduren serialisiert werden. Zudem führen Sprünge aus einer Parallelverzweigung heraus und in eine Parallelverzweigung hinein zu unsicheren bzw. verklemmenden SFC [6]. Bei SC sollen Ereignisse innerhalb eines SC vermieden werden, die in orthogonalen Komponenten erkannt werden müssen. Diese können zu ungewollten Endlosschleifen führen [72].

Da Prozeduren häufig aus vielen einzelnen Instanzen bestehen, ist es aus Gründen der Übersichtlichkeit möglich Hierarchieebenen einzufügen. Diese werden beispielsweise als Makroschritte oder Schwimmbahnen bezeichnet. Kritisch sind Hierarchieebenen jedoch, wenn die Abhängigkeiten zwischen den Ebenen nicht eindeutig spezifiziert sind. Beispielsweise führen die Definitionslücken bei den SFC zu einem implementationsspezifischen Verhalten und einer fehlenden Übertragbarkeit der Prozeduren auf andere Systeme [157].

Abstraktions- und Zuordnungsmodell

In den meisten Sprachen wird der Entwurfsprozess nicht durch ein Abstraktions- und Zuordnungsmodell unterstützt. Zu erwähnende Konzepte sind hier die Wiederverwendung von Grafcets als Einschließung, die Modellierung abstrakter Prozeduren in BPMN und das Rezeptmodell der IEC 61512.

Außerdem können verschiedene Sprachen unter bestimmten Rahmenbedingungen ineinander überführt werden, so dass die Spezifikation und die Implementierung in unterschiedlichen Sprachen erfolgen kann. Beispielsweise zeigt [158] die Überführung von Grafcet nach SFC. In BPMN wird eine Untermenge von Konstrukten spezifiziert, die mit BPEL kompatibel ist [83, 101]. EPK lassen sich in BPMN transformieren [131].

Aktions- und Aktivitätenmodell

Jede Prozedurbeschreibungssprache definiert ihre Interaktionsmöglichkeiten mit der Umgebung. In der Interaktionsmöglichkeit der Prozedur mit ihrer Umgebung unterscheiden sich die Beschreibungssprachen maßgeblich. Von der natürlichsprachlichen Beschreibung in EPK bis hin zur Beschreibung mit Programmiersprachen in SFC, von dem Setzen einzelner Signale in SIPN bis hin zu Dienstaufrufen in Grafchart kann die Interaktion auf verschiedenste Weisen erfolgen. Ein Grund hierfür ist im Unterschied zwischen Mensch

und Maschine hinsichtlich der Interpretationsfähigkeit informaler Aufrufe begründet. Ein weiterer Aspekt stellt der fortschreitende Einfluss der Informationstechnik dar, der beispielsweise die Nutzung eines Dienstsystems erst möglich macht.

Ausführungssteuerungsmodell

Die Prozedurbeschreibung ist eng mit Regeln zur operativen Ausführung der beschriebenen Prozeduren verknüpft. Daher enthält jede Prozedurbeschreibungssprache ein Konzept zur Ausführung der Prozedur. Dieses Konzept ist jedoch nicht immer vollständig spezifiziert. Teilweise hängt die Ausführung sogar von der Implementierung der Sprache (z. B. bei SFC) ab [6]. Ein weiteres Beispiel hierfür ist das dynamische Verhalten von Grafcet, welches nicht eindeutig definiert ist, mit anderen Worten, es liegt kein eindeutiges Modell vor, dass unabhängig von der Implementierung durch Rechner interpretiert werden kann [158].

4. Referenzmodell

Die Analyse von vierzehn Prozedurbeschreibungssprachen hat gezeigt, dass diese Sprachen alle einen gemeinsamen Kern haben. In diesem Kapitel wird ein Referenzmodell zur Prozedurbeschreibung entwickelt, das genau diesen gemeinsamen Kern abbildet. Dieses Referenzmodell, welches ein Meta-Modell (M2-Ebene) nach Kapitel 2.6.2, S. 20, ist, wird die grundlegenden Konzepte zusammenfassen, die in den Struktur- und Laufzeitinformationen der domänenspezifischen Sprachen enthalten sind. Die auf Basis des Referenzmodells erstellten Prozedurbeschreibungen sind Modelle (M1-Ebene). Im Folgenden werden zunächst die an ein solches Referenzmodell gestellten Anforderungen erhoben. Im Anschluss wird das Referenzmodell basierend auf den Elementen einer Prozedurbeschreibung (vgl. Kapitel 3.1, S. 24) unter Berücksichtigung der Anforderungen beschrieben. Das Referenzmodell besitzt zwei Darstellungsformen, eine grafische Notation und eine Darstellung als XML-Datei, die anschließend erläutert werden. Vor der Beschreibung der prototypischen Implementierung erfolgt die Formulierung der Anforderungen an die Ausführungseinheiten und an die Kommunikation, die sich durch das Referenzmodell ergeben.

4.1. Anforderungen an das Referenzmodell

Die Anforderungen an das Referenzmodell können zunächst in funktionale und nichtfunktionale Anforderungen[1] unterteilt werden [59]. Das Abdecken des gemeinsamen Kerns der in Kapitel 3, S. 24, vorgestellten Beschreibungssprachen ist eine funktionale Anforderung, die an das Referenzmodell gestellt wird. Das Referenzmodell hat den Anspruch, jede Prozedur beschreiben zu können. Es muss insbesondere einen Weg aufzeigen die unterschiedlichen Interaktionsmöglichkeiten der Prozedur mit ihrer Umgebung in einem Modell zu beschreiben. Auch eine durchgängige Darstellung von Hierarchien und Nebenläufigkeiten ist notwendig. Des Weiteren muss eindeutig feststehen, wie eine beschriebene Prozedur ausgeführt wird. Dies bezieht insbesondere das Verhalten der Prozedur im Fehlerfall mit ein.

Daneben existieren verschiedene nichtfunktionale Anforderungen, die die Anwendung des Referenzmodells vereinfachen. Da verschiedene Rollen (Modellierer, Programmierer und Anwender) mit einer Prozedurbeschreibung arbeiten müssen, ist eine einfache und intuitive Handhabung des Modells wichtig. Auf diese Weise ist eine Nachvollziehbarkeit der spezifizierten Abläufe möglich [62]. Zudem muss sich ein Prozedurmodell in domänenspezifische Sprachen überführen lassen, mit denen die Konfiguration, die Parametrierung und die Programmierung von Automatisierungslösungen durchgeführt werden [182]. Das Referenzmodell liefert demnach einen Beitrag zur Vereinfachung des Entwurfsprozesses einer Prozedur.

[1] Details zum Unterschied zwischen funktionalen und nichtfunktionalen Anforderungen können z. B. [97] entnommen werden.

Auch die vertikale Integration, d. h. die ebenenübergreifende Kommunikation innerhalb der Automatisierungspyramide, muss durch das Referenzmodell unterstützt werden. Aufgrund der hohen Komplexität ebenenübergreifender Prozedurmodelle ist eine Hierarchisierung unabdingbar [182]. Eine Kapselung der Informationen der Ausführungseinheiten in unterschiedliche Zustände ist eine Möglichkeit die Komplexität der unteren Ebenen der Automatisierungspyramide vor den oberen Ebenen zu verbergen. Diese zustandsbasierte Prozessführung entspricht den Überlegungen des Arbeitskreises „Modulare Anlagen“ der Interessengemeinschaft Automatisierungstechnik der Prozessindustrie (NAMUR) zur Steuerung modularer Prozessanlagen [134]. Die Kapselung ermöglicht zudem einen möglichst einfachen Einblick aus der ERP-Ebene in die aktuelle Produktion, wie in [69] gefordert. Ziel ist es die Verknüpfung der Produktionsprozesse mit den Lieferanten und den Kunden durch ein solches Referenzmodell zu erleichtern [69, 150]. Ein weiterer Vorteil ist die Wiederverwertung der erzeugten Lösungen, so dass die Fehleranfälligkeit im Entwurf und der Aufwand bei einer erneuten Nutzung sinken [59].

Das Referenzmodell muss sich in eine Industrie 4.0-Landschaft [125] einbinden lassen und sich dabei insbesondere in das Referenzarchitekturmodell Industrie 4.0 (RAMI) eingliedern [2]. Des Weiteren soll das Referenzmodell in der Lage sein, mit Industrie 4.0-Komponenten [53] zu interagieren. Eine der Anforderungen, die Cyber-Physical Systems (CPS) an die Automatisierungslösung stellen, sind Änderungsmöglichkeiten der Anwendungsfunktionen im operativen Betrieb [182]. Dies beinhaltet insbesondere eine Erweiterung des Funktionsumfangs [131], die Verteilung von Funktionen in verteilten Systemen sowie die Adaption und Rekonfiguration operativer Führungsfunktionen [52]. Somit muss auch das Referenzmodell diesen Anforderungen an die Flexibilität genügen.

Prozedurmodelle sollen mit Hilfe des Referenzmodells einfach erstellt, eindeutig interpretiert und während der Ausführung leicht verstanden werden können [152]. Allerdings wird in dieser Arbeit keine arbeitswissenschaftliche Analyse durchgeführt, wie sie z. B. in [131] ausgeführt wird.

Andere nichtfunktionale Anforderungen, die an eine Automatisierungslösung gestellt werden, liegen außerhalb des Referenzmodells. Die Sicherstellung des Determininismus und der Echtzeitfähigkeit der Kommunikation (vgl. [182]) zwischen Steuerung und Ausführungseinheit ist Aufgabe des verwendeten Kommunikationsmittels, welches nicht durch das Referenzmodell festgelegt wird. Dies gilt ebenso für den Schutz vor unberechtigten Zugriffen. Die Erfüllung der funktionalen Sicherheit (vgl. [182]) ist ebenso die Aufgabe der Ausführungseinheiten wie das Ablehnen von Belegungsmaßnahmen, die im aktuellen Zustand nicht erlaubt sind (vgl. [184]). Beides wird daher im Referenzmodell nicht behandelt. Ein Änderungsmanagement und eine Verifikation (vgl. [62]) der erstellten Prozedurmodelle ist hingegen Aufgabe der Modellierungswerkzeuge, die in der täglichen Praxis das Erstellen der Prozedurmodelle ermöglichen. Die Ausführungsumgebung muss die Anwendung des Referenzmodells in der Ausführungsphase durch Anlagenfahrer mit unterschiedlicher Qualifikation und durch Nutzergruppen mit unterschiedlichen Rechten (vgl. [182]) unterstützen.

4.2. Modellbeschreibung

Auch das hier vorgestellte Referenzmodell muss den Elementen einer Prozedurbeschreibungssprache (vgl. Kapitel 3.1, S. 24) Rechnung tragen. Daher werden die Elemente des

Referenzmodells in den folgenden Abschnitten vorgestellt und ihr Zusammenwirken mit den anderen Modellelementen erläutert. Die Modellbeschreibung basiert auf eigenen Arbeiten, die zu Teilen bereits in [130] und [152] veröffentlicht sind.

4.2.1. Aufbaumodell

Das Referenzmodell beinhaltet die Basiselemente Ausführungsrahmen (**ExecutionFrame**), Schritt (**Step**) und Transition (**Transition**), die in Abbildung 4.1 dargestellt sind.

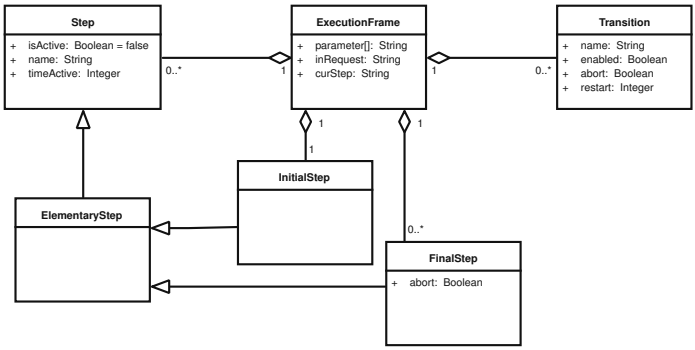

Abbildung 4.1.: Aufbaumodell des Referenzmodells in UML-Notation

Eine Prozedurbeschreibung stellt einen **ExecutionFrame** zur Verfügung, der einen eigenen Namensraum definiert. Dem Ausführungsrahmen können Parameter im Attribut *parameter* übergeben werden, die die Prozedurbeschreibung flexibel macht, beispielsweise hinsichtlich zu produzierender Mengen. Des Weiteren speichert ein Attribut *inRequest* den zuletzt eingegangenen Befehl. Das Attribut *curStep* enthält den zum aktuellen Zeitpunkt aktiven Schritt. Die beiden letztgenannten Attribute sind nur während der operativen Ausführung der Prozedur interessant.

Ein Schritt ist hierbei in Anlehnung an die Definition in [34] ein Beharrungszustand der Prozedur. Die Klasse **Step** besitzt einen Namen *name*, der Auskunft über die Funktion des Schritts geben kann. Ein **Step** kann aktiv oder inaktiv sein. Dies ist in einem Attribut *isActive* gespeichert. Weiterhin ist die Zeitspanne der Aktivität eines Schrittes von Interesse. Diese ist in der Variablen *timeActive* hinterlegt. Die Angabe *timeActive* wird in der Einheit *ms* angegeben. Die Attribute *isActive* und *timeActive* sind ebenfalls nur während der operativen Ausführung der Prozedur von Interesse.

Die Klasse Elementarschritt (**ElementaryStep**) ist von der Klasse **Step** abgeleitet. Der Anfangsschritt (**InitialStep**) und der Endschritt (**FinalStep**) erben wiederum von der Klasse **ElementaryStep**. Unterhalb eines Ausführungsrahmens existiert genau ein Startschritt. Daraus folgt, dass in einem Prozedurmodell mindestens ein Schritt enthalten sein muss. Die Anzahl der Endschritte ist nicht vorgegeben, sondern anwendungsfallabhängig.

Zyklische Prozeduren müssen keinen Endschritt haben, während terminierende Prozeduren einen oder mehrere Endschritte haben. Das Attribut *abort* der Klasse **FinalStep** wird für die Ausführungssteuerung (vgl. Kapitel 4.2.5, S. 78) benötigt.

In Anlehnung an [34] ist eine Transition ein Übergangselement, das den Wechsel zwischen zwei Schritten steuert. Die Klasse **Transition** besitzt ebenfalls einen Namen *Name*, das Attribut *enabled* gibt an, ob die Transition ausgewertet wird. Die Attribute *abort* und *restart* der Klasse **Transition** werden für die Ausführungssteuerung (vgl. Kapitel 4.2.5, S. 78) benötigt. Die Transitionen sind ebenfalls unter dem Ausführungsrahmen angeordnet. Die Menge der Transitionen kann leer sein, wenn der Startschritt der einzige Schritt der Prozedur ist.

4.2.2. Hierarchie- und Vernetzungsmodell

Innerhalb des Ausführungsrahmens werden Schritte und Transitionen alternierend durch Kanten miteinander verknüpft (vgl. Abbildung 4.2). Eine Transition hat genau einen Vorgänger- und genau einen Folgeschritt (vgl. die Assoziationen *preStep* und *sucStep* in Abbildung 4.2). Alle Kanten haben Kantengewicht Eins, so dass eine Prozedurbeschreibung ein ungewichteter Graph ist. Da eine Kante immer einen Schritt und eine Transition verbindet, folgt zudem, dass der Graph bipartit ist.

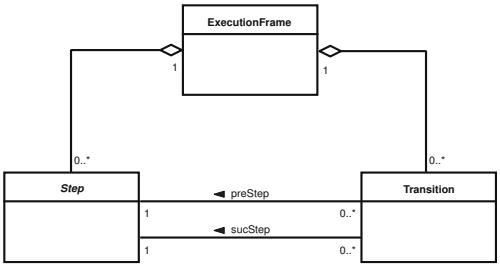

Abbildung 4.2.: Vernetzungsmodell des Referenzmodells in UML-Notation

Die Basiselemente des Aufbaumodells zusammen mit den Kanten des Hierarchie- und Vernetzungsmodells sind ausreichend zur Definition linearer terminierender und linearer zyklischer Prozeduren. Auch Alternativverzweigungen können mit diesen Elementen modelliert werden, da ein Schritt mehrere eingehende und ausgehende Transitionen haben kann. Daher muss es kein separates Element für Alternativverzweigungen und -vereinigungen geben. Bei einer Alternativverzweigung können mehrere Transitionsbedingungen gleichzeitig *wahr* werden. Deshalb ist ein deterministisches Entscheidungskriterium notwendig. Das Referenzmodell fordert den gegenseitigen Ausschluss der Transitionsbedingungen, da sich die grafische Priorisierung und die explizite Angabe der Priorität (welches weitere Möglichkeiten bei den in Kapitel 3, S. 24, betrachteten Sprachen sind) darauf abbilden[2] lassen.

[2]Die Abbildung ist in Anhang C.2, S. 121, dargestellt.

Allerdings werden derart erzeugte Prozeduren schnell sehr unübersichtlich. Daher ist es notwendig, dass das Referenzmodell den Aufbau von Hierarchien unterstützt. Dies erfolgt mit Hilfe der Klasse Makroschritt (**MacroStep**), die in Abbildung 4.3 dargestellt ist. Ein Makroschritt ist eine logische Gruppierung von Schritten, Transitionen und Kanten. Er nutzt denselben Adressraum wie der umgebende Schritt. Da Makroschritte spezielle Schritte sind, kann ein Makroschritt sowohl Vorgänger- als auch Folgeschritt einer Transition sein. In einem Makroschritt muss genau ein Anfangs- und genau ein Endschritt vorhanden sein. Des Weiteren ist es möglich, innerhalb einer Prozedur andere Prozeduren aufzurufen. Diese haben einen eigenen Ausführungsrahmen und einen eigenen Adressraum. Sie sind somit im Gegensatz zu Makroschritten unabhängig von der aufrufenden Prozedur.

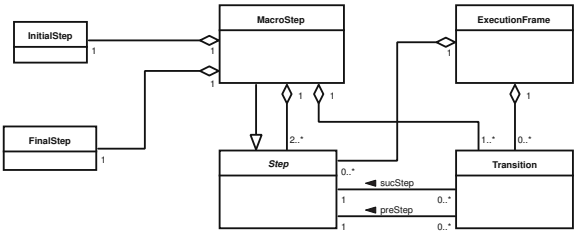

Abbildung 4.3.: Makroschritt des Referenzmodells in UML-Notation

Die Modellierung nebenläufiger Vorgänge ist, wie bereits im Vergleich der analysierten Sprachen (vgl. Kapitel 3.5, S. 64) beschrieben, problematisch, da auf diese Weise Verklemmungen und Unsicherheiten entstehen können. Einige Ansätze (vgl. z. B. [196]) gehen daher dazu über, solche nebenläufigen Abläufe zu verbieten. Für das Referenzmodell sind die nebenläufigen Abläufe allerdings zwingend erforderlich, da diese in den meisten Sprachen enthalten und gerade bei Prozeduren zur Steuerung von Geschäftsprozessen auch essentiell sind. Im Referenzmodell sind zwei Möglichkeiten zur Modellierung vorgesehen, die Verwendung eines P-Makroschritts (**PMacroStep**) und der parallele Aufruf mehrerer unabhängiger Prozeduren:

- Ein P-Makroschritt wird verwendet, damit die Nebenläufigkeit gekapselt wird (vgl. Abbildung 4.4).

  Auf diese Weise ist zunächst sichergestellt, dass die aufrufende Prozedur zu jeder Zeit in einem definierten Schritt ist. Wie „normale“ Makroschritte auch, besitzt ein P-Makroschritt einen Anfangs- und einen Endschritt. Der Anfangsschritt wird über eine Verzweigung (**Fork**) mit mindestens zwei Schritten verbunden. Die Verbindung zwischen Anfangsschritt und Verzweigung erfolgt über die Assoziation *inFork*, diejenige zwischen der Verzweigung und den nachfolgenden Schritten über die Assoziation *outFork*. Die nebenläufigen Pfade werden über eine Zusammenführung (**Join**) mit dem Endschritt des P-Makroschritts zusammengeführt. Die Verbindung zwischen den letzten Schritten der jeweiligen nebenläufigen Pfade und der Zusammenführung erfolgt über die Assoziation *inJoin*, diejenige zwischen der Zusammenführung und dem Endschritt über die Assoziation *outJoin*. Verzweigung und Zusammenführung

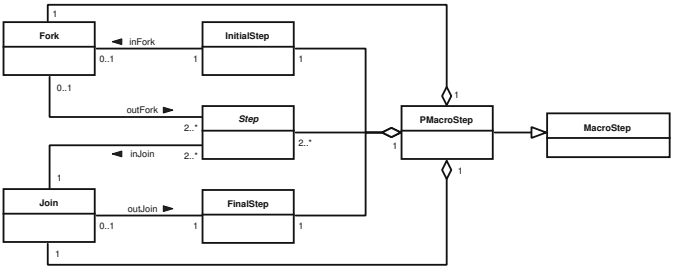

Abbildung 4.4.: P-Makroschritt des Referenzmodells in UML-Notation

sind damit ebenso wie eine Transition Übergangselemente. Weil eine Transition genau einen Vorgängerschritt und genau einen Nachfolgeschritt besitzen muss, können sie jedoch nicht von der Klasse Transition erben. Da ein P-Makroschritt ausschließlich über den Anfangsschritt betreten und über den Endschritt verlassen werden kann, wird eine Verklemmung oder eine Unsicherheit vermieden[3]. In Abbildung 4.5 ist ein Beispiel mit zwei nebenläufigen Pfaden skizziert, wie die entsprechenden Instanzen und Assoziationen in einem P-Makroschritt verwendet werden.

- Werden mehrere Prozeduren parallel aufgerufen, laufen diese unabhängig voneinander ab. Sollte eine Synchronisation zwischen den Prozeduren oder zwischen aufgerufenen Prozeduren und aufrufender Prozedur gewollt sein, muss diese explizit projektiert werden.

[3]Dies gilt nur in Hinblick auf den von PN und SFC bekannten Verlust bzw. das Entstehen von Marken. Durch ungünstigen Aufruf von Aktionen kann in einem Schritt eine Aktion gestartet werden, die darin resultiert, dass eine Transition nicht mehr feuern kann.

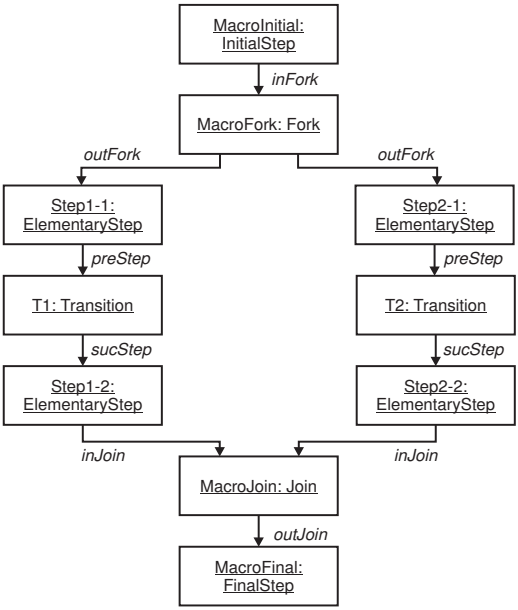

Abbildung 4.5.: Beispiel für einen P-Makroschritt

4.2.3. Aktions- und Aktivitätenmodell

Im Aktions- und Aktivitätenmodell ist die Anbindung der Prozedur an die Umgebung beschrieben. Dies betrifft sowohl die Einwirkung der Prozedur auf andere Steuerungen und Ausführungseinheiten als auch die Wirkung der Umgebung auf die Prozedur. Die in Kapitel 3, S. 24, untersuchten Beschreibungssprachen benutzen dazu meist Signale. Dies steht im Widerspruch zu der flexiblen Integration in eine Industrie 4.0-Umgebung (vgl. Kapitel 4.1, S. 67).

Einwirkung der Prozedur auf die Umgebung

Zur Einwirkung der Prozedur auf die Umgebung nutzt das Referenzmodell die Klasse Aktion (**Action**). Aktionen sind immer einem Elementarschritt zugeordnet, eine Zuordnung zu einem Makroschritt ist nicht möglich. **Action** ist eine abstrakte Klasse, von der die Klassen Dienstaufruf (**ServiceCall**) (vgl. Kapitel 2.5.1, S. 16) und Prozeduraufruf (**ProcedureCall**) erben (siehe Abbildung 4.6). Ein Dienstaufruf kann z. B. das Setzen einer Variablen initiieren, aber auch eine mündliche Anordnung eines Chefs an seinen Mitarbeiter sein. Dienstaufrufe sind immer zielgebunden, ein Multicast oder ein Broadcast sind nicht vorgesehen. Ein Prozeduraufruf sendet einen Befehl an eine Prozedur. Konkret bedeutet dies, dass die Attribute *inRequest* und *parameter* des Ausführungsrahmens einer Prozedur gesetzt werden. Diese können anschließend von der Empfängerprozedur oder von der zugehörigen Ausführungssteuerung interpretiert werden.

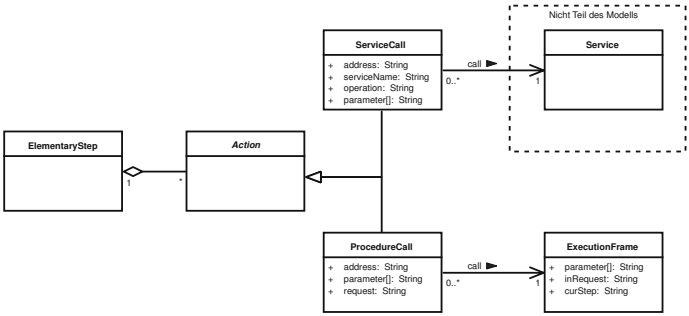

Abbildung 4.6.: Aktionsmodell des Referenzmodells in UML-Notation

Bei einem Dienstaufruf werden die Adresse *address* des Dienstanbieters, der aufgerufene Dienstname *serviceName*, die konkrete Operation *operation* sowie optionale Parameter *parameter* benötigt. Jeder Dienstaufruf ist genau einem elementaren Schritt zugeordnet, allerdings kann ein Dienst (**Service**) mehrfach aufgerufen werden. Die Dienste werden durch die Ausführungseinheiten angeboten, an die dadurch bestimmte Anforderungen gestellt werden (vgl. Kapitel 4.4, S. 85). Die Prozedur steuert demnach die Ausführungseinheiten nur noch durch die Dienstaufrufe an. Der Namensraum legt hierbei den Ausgangspunkt der Adressierung der Ausführungseinheiten fest. Dieses Konzept gliedert sich in das kybernetische Grundprinzip (vgl. Kapitel 2.2, S. 7) ein, da auf diese Weise eine ideale Trennung

zwischen Steuerndem und Gesteuertem vorliegt. Die Steuerung durch Dienstaufrufe wird auch als Dienstorchestrierung bezeichnet.

Dienstaufrufe sind Aktionen und somit technologisch zeitlose Vorgänge. Die Ausführung des Dienstes ist allerdings eine Aktivität, die eine bestimmte Dauer hat. Da Dienstaufrufe nicht auf einen Rückgabewert warten, ist diese Dauer für die Prozedur nicht relevant. Falls die Fertigstellung des Dienstes durch die Prozedur überwacht werden soll, muss eine explizite Zustandsabfrage des Dienstanbieters in einer Transitionsbedingung erfolgen. Dies ist ratsam, da Ausführungseinheiten Dienstanfragen auch ablehnen können (vgl. [47]). Das Referenzmodell ist unabhängig von der zugrunde liegenden Kommunikationsplattform und des Dienstsystems. Zur Nutzung des Modells muss jedoch ein Kommunikationssystem ausgewählt werden. Weiter macht ein solches dienstaufrufbasiertes Aktionsmodell die Definition einer einheitlichen Aufrufnotation der Dienste notwendig (vgl. z. B. [47, 77, 184]).

Wie bereits erwähnt, können neben Dienstaufrufen auch Prozeduraufrufe als Aktion verwendet werden. Beim Aufrufen einer Prozedur muss neben dem zu übergebenden Befehl *request* die Adresse *address* des Ausführungsrahmens der Prozedur bekannt sein. Hier besteht ebenfalls optional die Möglichkeit Parameter *parameter* zu definieren.

Einwirkung der Umgebung auf die Prozedur

Die **Transition** ist das Element, welches den Wechsel des aktiven Schritts in Reaktion auf den Umgebungszustand steuert. Der Übergang ist durch eine Übergangsbedingung (**Condition**) beschrieben, die der Transition zugeordnet wird. Der Übergang erfolgt unter der Nebenbedingung, dass die Transition freigegeben ist. Eine Übergangsbedingung besteht aus logischen Termen (**LogicalTerm**), wie in Abbildung 4.7 dargestellt ist.

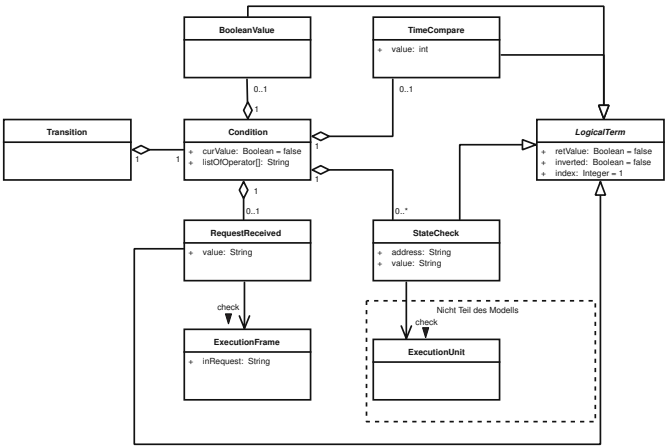

Abbildung 4.7.: Transitionsbedingungen des Referenzmodells in UML-Notation

Jeder logische Term liefert einen Rückgabewert *retValue* an die Bedingung zurück. Die Rückgabewerte der einzelnen Bedingungen sind über die in der Liste *listOfOperator* ent-

haltenen booleschen Operatoren verknüpft und bilden so den Rückgabewert *curValue* der Übergangsbedingung. Das Attribut *inverted* eines **LogicalTerm**s gibt an, ob der Rückgabewert invertiert werden soll. Die Zuordnung zwischen den logischen Termen und der Liste *listOfOperator* erfolgt über das Attribut *index*. Die logischen Terme mit dem Index 1 und dem Index 2 sind durch den Operator im ersten Eintrag der Liste verknüpft usw.

Von der abstrakten Klasse **LogicalTerm** sind die Klassen boolescher Wert (**BooleanValue**), Zeitvergleich (**TimeCompare**), Befehl eingegangen (**RequestReceived**) und Zustandsüberprüfung (**StateCheck**) abgeleitet (vgl. Abbildung 4.7):

- Ein boolescher Wert gibt immer den Rückgabewert *wahr* zurück. Dieser wird zur Modellierung selbstterminierender Schritte benötigt, bei denen der Schrittwechsel ohne zugehörige Bedingung erfolgt.
- Mit Hilfe des Zeitvergleichs wird das Attribut *timeActive* des Schritts vor der Transition mit dem in *value* (Angabe in *ms*) gespeicherten Wert verglichen. Ist der Schritt länger aktiv als die Vorgabe in der Transition, liefert der Zeitvergleich *wahr* an die Übergangsbedingung zurück.
- Die Klasse **RequestReceived** überprüft das Attribut *inRequest* des überlagerten Ausführungsrahmens. Stimmt dieser mit dem Wert des Attributs *value* überein, wird *retValue* auf *wahr* gesetzt. Dies ist auch möglich, wenn die Abfrage innerhalb eines Makroschritts erfolgt.
- Eine Zustandsüberprüfung kann den aktuellen Zustand einer Ausführungseinheit abfragen. Stimmt die Abfrage des Attributwerts an der Adresse *address* mit dem in *value* gespeicherten Wert überein, wird der *retValue* der Zustandsüberprüfung auf *wahr* gesetzt.

Die Klassen Verzweigung (**Fork**) und Zusammenführung (**Join**) (vgl. Abbildung 4.4) sind spezielle Übergangsbedingungen. Bei den beiden Klassen wird das Weiterschalten nicht durch Transitionsbedingungen gesteuert. Vielmehr schaltet die Verzweigung direkt, wenn der Schritt „MacroInit“ aktiviert worden ist. Die Zusammenführung aktiviert den Schritt „MacroFinal“, sobald alle Vorgängerschritte aktiv sind.

4.2.4. Abstraktions- und Zuordnungsmodell

Das Abstraktions- und Zuordnungsmodell beschreibt Konzepte, die während des Entwurfsprozesses einer Prozedur nützlich sind. Die Analyse der Sprachen in Kapitel 3, S. 24, hat gezeigt, dass der Entwurfsprozess anders als derjenige in der Funktionsbausteintechnik abläuft. Gemeinsam haben beide Entwurfsprozesse eine funktionale und organisatorische Trennung in Entwicklung, Engineering und Laufzeit. Funktionsbausteinnetze werden nach dem Typ-Instanz-Konzept erzeugt. Prozeduren hingegen werden zu Beginn des Engineering-Prozesses abstrakt definiert und immer weiter konkretisiert. Der finale Konkretisierungsgrad ist abhängig von dem verwendeten Steuerungssystem (vgl. Kapitel 2.2.3, S. 10). Mit anderen Worten, das Abstraktions- und Zuordnungsmodell legt auch die Möglichkeiten der Flexibilität fest. Im Referenzmodell sind zwei Möglichkeiten zur Erhöhung

der Flexibilität vorgesehen, die Verwendung von Rollen einerseits sowie typbezogene Prozedurbeschreibungen andererseits.

Anstelle eines Dienstaufrufs an eine konkrete Ausführungseinheit (**ExecutionUnit**) kann im Referenzmodell der Aufruf durch eine Rolle (**Role**) abstrahiert werden (vgl. Abbildung 4.8).

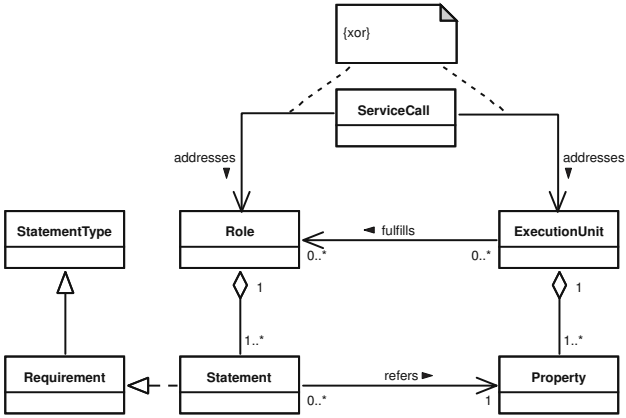

Abbildung 4.8.: Verwendung von Rollen im Referenzmodell

Rollen sind ein bewährtes Konzept zur Abstraktion zwischen Funktionalität und technischer Realisierung[4] bzw. zwischen einem Menschen und der Aufgabe, die er zu erfüllen hat (vgl. z. B. [40, 49, 57, 112]). Die Umsetzung des Rollenkonzepts kann durch ein Merkmalsystem erfolgen (vgl. Kapitel 2.6.2, S. 22). In der Rolle werden Aussagen (**Statement**) vom Typ Anforderung (**Requirement**)[5] der Prozedur über die Ausführungseinheiten beschrieben. Diese können im Laufe des Entwurfsprozesses weiter spezifiziert oder durch eine konkrete Ausführungseinheit ersetzt werden. Des Weiteren ist es möglich die Zuweisung erst während der Ausführung der Prozedur vorzunehmen. Dies gewährleistet die geforderte Flexibilität des Referenzmodells. Das Zusammenspiel zwischen Merkmalen (**Property**) und Anforderungen während der Entwicklung einer Automatisierungslösung wird in [57] als eine Möglichkeit der Vereinfachung des Entwurfsprozesses gesehen. In [59] wird die Bedeutung von Merkmalen für den Entwurfsprozess ebenfalls betont.

Im Regelfall werden Prozeduren zur Steuerung einer konkreten Ausführungseinheit entworfen, da gerade im Umfeld der chemischen Industrie Anlagen häufig Einzellösungen sind. Dies wird als singuläre Prozedurfestlegung bezeichnet (Abbildung 4.9 links). Eine typbezogene Prozedurbeschreibung ist dann sinnvoll, wenn die Prozedur mehrfach in leicht

[4]Das Prinzip der Rollen ist aus dem Rohrleitungs- und Instumentrierungsdiagramm (R&I) bekannt. Die grafischen Symbole des R&I sind lediglich Platzhalter für die technische Realisierung [40].

[5]Nach [12] sind z. B. Materialanforderungen, Equipment-Informationen, Asset-Informationen und Personalinformationen zu berücksichtigen.

veränderter Form verwendet werden soll. Dies ist z. B. bei Package Units oder bei modularen Anlagen der Fall (Abbildung 4.9 rechts). Ein weiterer Anwendungsfall ist die Nutzung einer Prozedur in mehreren parallelen Fertigungsstraßen.

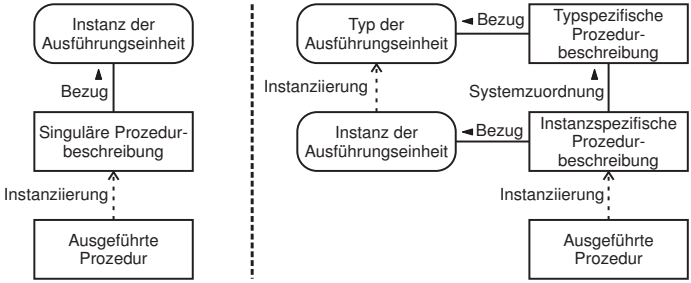

Abbildung 4.9.: Singuläre (links) und typbezogene (rechts) Prozedurfestlegung

In diesem Fall ist die Prozedur schon für den generischen Ausführungseinheitstypen zu entwerfen. Die Instanzbildung der Ausführungseinheitstypen entspricht der kundenspezifischen Konfiguration der Typen. Analog dazu wird der Prozedurtyp instanziiert und auf die kundenspezifische Konfiguration angepasst. Hinsichtlich der Wiederverwendbarkeit und dem Nutzen von Prozedurbibliotheken bietet dieses Vorgehen einen Vorteil.

4.2.5. Ausführungssteuerungsmodell

Bei der Ausführung einer Prozedur muss sowohl der Regelablauf als auch die Reaktion auf Fehler berücksichtigt werden (vgl. z. B. [137, 171]). Das Referenzmodell besitzt einen Zustandsautomaten, der die Prozedur im vollautomatischen Regelablauf und im Fehlerzustand steuert (vgl. Abbildung 4.10).

Abbildung 4.10.: Ausführungssteuerungsmodell des Referenzmodells in UML-Notation

Der Zustandsautomat ist in Abbildung 4.11 dargestellt.

Eine Prozedur kann *inaktiv* (**Inactive**) oder *aktiv* (**Active**) sein. Der Transitionsübergang ist zeitlos. Auf diese Weise ist sichergestellt, dass zu jeder Zeit in einer aktiven Prozedur genau ein Schritt aktiv ist. Die grobe Klassifikation in aktiv und inaktiv lässt sich ferner in sechs Zustände verfeinern: *Ruhend* (**Idle**), *Abgebrochen* (**Aborted**), *Laufend* (**Running**), *Schritt haltend* (**Step Hold**), *Abbrechend* (**Aborting**) und *Neustartend*

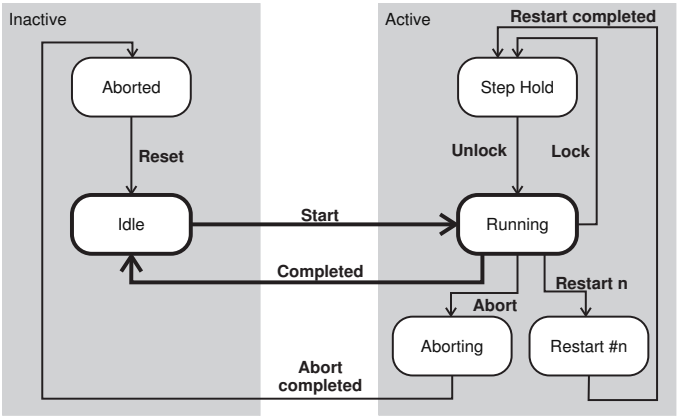

Abbildung 4.11.: Zustandsautomat einer Prozedur

(**Restart n**). Es kann mehrere **Restart**-Zustände geben. Dies ist durch das „n“ in Abbildung 4.11 angedeutet.

Idle

Wird eine Prozedur geladen, ist sie zunächst inaktiv und befindet sich im Zustand **Idle**. In einer inaktiven Prozedur ist kein Schritt aktiv und daher sind alle Transitionen ebenfalls nicht freigeschaltet. Sobald an die Prozedur der Befehl `Start` gesendet wird, wird die Prozedur aktiviert und wechselt in den Zustand **Running**.

Running

Im Zustand **Running** beginnt der Ablauf der Prozedur im Normalfall:

1. Der Anfangsschritt wird aktiviert.
2. Anschließend wird die Variable *timeActive* auf Null gesetzt und fortlaufend aktualisiert. Zudem werden alle in diesem Schritt definierten Aktionen einmalig ausgeführt.
3. Nach der Aktionsausführung[6] werden alle Transitionen mit $abort = false$ und $restart = 0$, die an diesen Schritt anschließen, freigeschaltet und die Übergangsbedingungen ausgewertet[7].
4. Ist ein Rückgabewert einer Übergangsbedingung *wahr*, feuert die entsprechende Transition. Da die Transitionsbedingungen einander ausschließen müssen, kann maximal eine Transition feuern.

[6] Im Falle einer SPS-Steuerung erfolgt dieser Schritt im nächsten Zyklus.
[7] Ist eine Transition nicht freigeschaltet, so wird die Bedingung nicht ausgewertet und die Bedingung liefert *falsch* als Rückgabewert.

5. Wenn eine Transition feuert, wird der Folgeschritt aktiviert und der Vorgängerschritt deaktiviert. Somit sind auch die Transitionen nach dem Vorgängerschritt nicht mehr freigeschaltet.

6. Der Folgeschritt kann ein elementarer Schritt, ein gewöhnlicher Makroschritt, ein paralleler Makroschritt oder ein Endschritt sein.
   a) Bei einem elementaren Schritt setzt der Ablauf mit Punkt 2 fort.
   b) Die Aktivierung eines innenliegenden Anfangsschritts erfolgt, wenn ein Makroschritt aktiviert wird. Die Ausführung innerhalb des Makroschritts erfolgt ab Punkt 1. Die Transition nach dem Makroschritt wird erst freigeschaltet, wenn der Makroschritt seinen Endschritt erreicht hat. Der Ablauf setzt dann mit Punkt 3 fort. Ein Deaktivieren eines Makroschritts führt auch zur Deaktivierung des Endschritts innerhalb des Makroschritts.
   c) Die Ausführung eines parallelen Makroschritts erfolgt analog zu der eines gewöhnlichen Makroschritts. Zu beachten ist, dass die Ausführungsreihenfolge der nebenläufigen Pfade vor der Ausführung festgelegt worden sein muss.
   d) Ein Endschritt sendet implizit den Befehl `Completed` an den Zustandsautomaten. Dieser Befehl bewirkt, dass die Prozedur inaktiv wird und in den Zustand **Idle** wechselt.

Der Vorgang der Aktionsausführung und des Weiterschaltens wird demnach solange wiederholt, bis ein Endschritt erreicht ist. Im Falle einer SPS als Steuerungssystem sei festgehalten, dass in einem Zyklus maximal eine Transition feuern kann. Führt ein Mensch die Prozedur aus, so kann er einen Schrittwechsel nur dann vollziehen, wenn er merkt, dass eine Transitionsbedingung *wahr* geworden ist. Die Geschwindigkeit der Ausführung der Prozedur hängt demnach maßgeblich von den kognitiven Fähigkeiten des Menschen sowie der grafischen Aufbereitung der Bedingungen ab.

Neben des Regelablaufs (fett gedruckte Linien in Abbildung 4.11) können auch Eingriffe in den Prozedurablauf vorgenommen werden. Beispiele hierfür sind

- Fehlermeldungen durch Ausführungseinheiten oder das Kommunikationssystem,
- Befehle durch überlagerte Führungsebenen,
- Zustandsänderungen aufgerufener Prozeduren,
- Zeitüberschreitungen der auszuführenden Dienste, z. B. durch Rohstoffmangel oder Krankheitsmeldung der entsprechenden Person, oder
- menschliche Eingriffe, z. B. durch Anlagenfahrer.

Die Analyse in Kapitel 3, S. 24, hat gezeigt, dass vielfach ungeklärt ist, wie andere Hierarchieebenen auf Fehler in einer Prozedur reagieren müssen. Für ein deterministisches Ausführungsverhalten ist dies jedoch unabdingbar. Daher wird im Folgenden explizit das Verhalten von Makroschritten in den jeweiligen Zuständen beschrieben. Aufgerufene Prozeduren sowie überlagerte Steuerungen sind in ihrer Ausführung unabhängig von der betrachteten Hauptprozedur. Daher müssen Beeinflussungen explizit über Aktionen in den

Abbruch- und Neustartprozeduren oder durch Zustandsabfragen in den betreffenden Prozeduren modelliert werden.

Manuelle Eingriffe müssen in den Transitionsbedingungen projektiert sein. Diese können z. B. über eine grafische Nutzeroberfläche an den Auftragseingang des Ausführungsrahmens geschickt werden. Hierzu zählt beispielsweise auch die halbautomatische Ausführung einer Prozedur. Die Eingriffe können je nach ihrer Schwere unterschiedliche Auswirkungen auf den Prozedurablauf haben. Die niedrigste Stufe eines Eingriffs ist das Halten. Auf dieser Stufe sind keine weiteren Vorkehrungen zum Fortsetzen des gesteuerten Prozesses zu treffen. Ein Neustart bewirkt die Ausführung von Aktionen, die zur Sicherung und Stabilisierung des Prozesses führen, so dass er an einer definierten Stelle fortgesetzt werden kann. Ein Abbruch führt den Prozess in einen sicheren Zustand.

Step Hold

Der Zustand *Schritt halten* (**Step Hold**) wird durch den Befehl `Lock` an den Zustandsautomaten erreicht. In diesem Zustand sind alle Transitionen gesperrt, d. h., der aktuelle Schritt wird so lange gehalten, bis ein Befehl `Unlock` eingeht. Dieser Befehl bewirkt einen Wechsel in den Zustand **Running**.

Ein (gewöhnlicher oder paralleler) Makroschritt besitzt denselben Ausführungsrahmen wie die ausführende Prozedur. Daher liegt derselbe Zustandsautomat zugrunde, d. h., auch innerhalb eines Makroschritts wird der aktuelle Schritt gehalten. Bei der Reaktivierung wird die Ausführung des Makroschritts an der gehaltenen Stelle fortgesetzt.

Restart

Für einen *Neustart* (**Restart**) müssen in der Prozedur entsprechende Logiken enthalten sein. Diese werden durch entsprechend ausgezeichnete Transitionen (vgl. das Attribut *restart* in Abbildung 4.6, S. 74) aktiviert, wenn ein Befehl `Restart n` empfangen wird. Da unterschiedliche Rücksprungpunkte möglich sind, können auch mehrere Neustart-Prozeduren durchgeführt werden. Diese werden durch den Parameter `n` des Befehls `restart` ausgewählt. Am Ende eines Neustarts muss explizit der Befehl `Restart completed` gesendet werden, der dazu führt, dass der Zustand *Step Hold* eingenommen wird. Innerhalb eines Makroschritts können keine Neustart-Abläufe enthalten sein. Es ist jedoch möglich innerhalb der Hauptprozedur eine Neustart-Transition an den Makroschritt anzubringen. Diese beendet den Makroschritt instantan, sobald sie ausgelöst wird.

Aborting

Für einen Abbruch müssen ebenfalls in der Prozedur entsprechende Logiken enthalten sein, die im Zustand *Abbrechend* (**Aborting**) ausgeführt werden. Die Aktivierung erfolgt durch entsprechend ausgezeichnete Transitionen (vgl. das Attribut *abort* in Abbildung 4.6, S. 74), wenn ein Befehl `Abort` empfangen wird. Bei einem Abbruch wird eine terminierende Kette aktiviert. Der Endschritt der terminierenden Kette sendet einen impliziten Befehl `Abort completed` an den Zustandsautomaten. Dieser bewirkt einen Übergang in den Zustand **Aborted**. Da ein Makroschritt maximal einen Endschritt haben darf, kann sich kein Abbruch innerhalb eines Makroschritts befinden. Es ist jedoch möglich, innerhalb der Hauptprozedur eine Abbruch-Transition an den Makroschritt anzubringen. Diese beendet den Makroschritt instantan, sobald die Abbruch-Transition ausgelöst wird.

Aborted

Der Wechsel in den Zustand *Abgebrochen* (**Aborted**) deaktiviert die Prozedur. Vor der Reaktivierung der Prozedur muss ein expliziter Befehl `Reset` von einer externen Quelle an die Prozedur gesendet werden. Dieser Befehl ändert den Zustand zu **Idle**. Der Zustand **Aborted** kann somit als Fehlerspeicher und der Befehl `Reset` als Quittierung interpretiert werden.

Zur Verringerung der visuellen Komplexität sei der Hinweis gegeben, dass die Neustart- und Abbruch-Prozedurschritte in Makroschritten oder in eigenen Prozeduren gekapselt und nur dort eingefügt werden sollen, wo es notwendig ist. Ein zweiter Hinweis gilt der erneuten Aktionsausführung, wenn eine Neustart-Kette wieder zu dem Schritt zurückgeführt wird, von dem der Neustart ausgeführt worden ist.

4.3. Darstellungsformen des Referenzmodells

Im bisherigen Verlauf dieses Kapitels ist das Referenzmodell als UML-Klassendiagramm betrachtet worden. Der Übersichtlichkeit halber sind lediglich Ausschnitte präsentiert worden, das vollständige Diagramm ist in Abbildung 6.1, S. 101 abgebildet. Die Darstellung als Klassendiagramm ist jedoch zu komplex für die tägliche Anwendung. Daher sind alternative Darstellungsformen notwendig.

4.3.1. Visualisierung

Die grafische Darstellung des Referenzmodells ist bisher noch nicht behandelt worden. Hierbei sind zwei Sichten auf das Modell wichtig, nämlich die des Entwicklers und die des Nutzers der Prozedur.

Entwicklungsprozesse in der Prozessautomation zeichnen sich durch eng verzahnte Abläufe aus, bei denen verschiedene Gewerke parallel arbeiten [58]. Ziel ist es die Gewerkeintegration und -durchgängigkeit zu erhöhen [179]. Ein Blick ausschließlich auf das Gewerk „Automatisierungstechnik“ zeigt bereits, dass auf den verschiedenen Ebenen der Automatisierungspyramide (vgl. Kapitel 2.4, S. 14) Menschen mit unterschiedlichen Fähigkeiten arbeiten [151].

Mit Hilfe eines meta-modellbasierten Visualisierungssystems (vgl. z. B. [86]) können mehrere Visualisierungsformen für das Referenzmodell entwickelt werden (vgl. Abbildung 4.12). Die Elemente im Referenzmodell bilden den gemeinsamen Kern der verschiedenen Prozedurbeschreibungssprachen aus Kapitel 3, S. 24, so dass es möglich ist für jedes Element seine Visualisierung in der jeweiligen grafischen Notation zu entwerfen.

In einem Visualisierungsmodell (**VisualizationModel**) sind zunächst einmal Grafik-Bibliotheken (**LibraryOfGraphicalElements**) enthalten. Diese Grafikbibliotheken bestehen wiederum aus Grafikelementen (**GraphicalElement**). Innerhalb einer Bibliothek muss jedes der zwölf Elemente (**ProcedureElement**) aus dem Referenzmodell (Schritt, Anfangsschritt, Endschritt, Makroschritt, P-Makroschritt, Transition, Kante, Aktion, Bedingung, Verzweigung, Zusammenführung, Ausführungsrahmen) durch maximal[8] eines dieser Grafikelemente repräsentiert werden. Abbildung 4.12 zeigt, dass das Referenzmodell mit

[8] Das Referenzmodell muss nicht vollständig durch das Visualisierungsmodell abgedeckt sein. Soll z. B. keine Nebenläufigkeit verwendet werden, wird kein Visualisierungselement für einen P-Makroschritt benötigt.

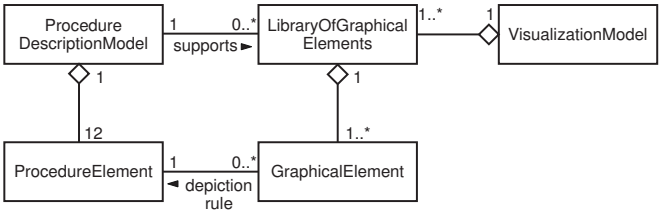

Abbildung 4.12.: Zusammenhang zwischen Referenzmodell und Visualisierung auf Meta-Modell-Ebene

mehreren Grafik-Bibliotheken eines Visualisierungsmodells verknüpft werden kann. Auf diese Weise ist es möglich für jede notwendige Prozedurbeschreibungssprache[9] eine eigene Grafik-Bibliothek zu erstellen. Der Entwicklungsingenieur kann sich die entsprechende Grafik-Bibliothek laden und in seiner gewohnten grafischen Umgebung planen (vgl. Abbildung 4.13). Greift ein anderer Entwickler auf die Prozedur zu, kann er eben diese Daten ohne jegliche Transformation nutzen. Er muss lediglich eine andere Grafik-Bibliothek verwenden.

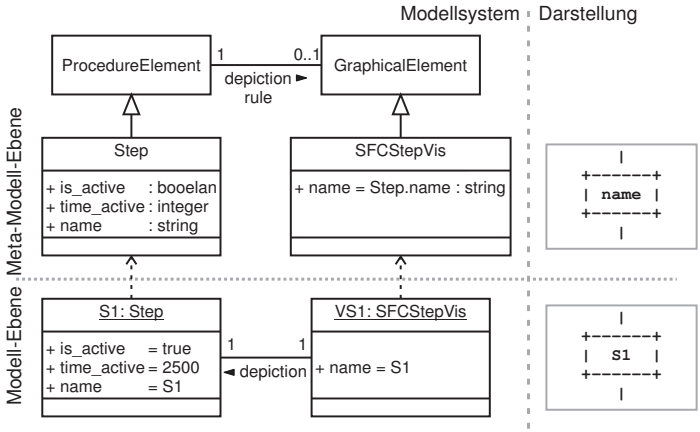

Abbildung 4.13.: Beispiel für die Instanzbildung, links im Modellsystem, rechts die Darstellung nach der IEC 61131-3

In Abbildung 4.13 ist die Anwendung der meta-modellbasierten Visualisierung beispielhaft an einem Schritt in SFC-Notation gezeigt. Wie in Abbildung 4.12 dargestellt, können

[9]Innerhalb eines Projekts wird typischerweise nur eine Teilmenge der Sprachen aus Kapitel 3, S. 24, verwendet.

einem Prozedurelement Grafikelemente zur Darstellung zugeordnet werden. Die Festlegung auf eine Bibliothek (hier SFC) bewirkt, dass es zum Prozedurelement **Step** genau ein Grafikelement „SFCStepVis“ gibt. Dieses besitzt eine Darstellung, wie in der IEC 61131-3 angegeben. Das Grafikobjekt besitzt einen Parameter, den Schrittnamen, der identisch mit dem Namen des zugehörigen Schrittes ist. Während des Entwurfsprozesses, d. h. der Erzeugung eines Prozedurmodells, wird das Prozedurelement Schritt mehrfach instanziiert. Zu jeder Instanz eines Schritts wird ebenfalls eine Instanz des zugehörigen Visualisierungsobjekts angelegt.

Auch für die operative Nutzung einer Prozedur ist es möglich Bibliotheken zu erstellen. Verschiedene Nutzergruppen (z. B. Handwerker, Anlagenfahrer oder ungelernte Kräfte) stellen aufgrund ihrer unterschiedlichen Fähigkeiten andere Anforderung an die Darstellung der Prozeduren, die sie als Anleitung für ihre Arbeit nutzen [138]. Anlagenfahrer sind beispielsweise nicht mit formalen Beschreibungsmitteln vertraut [158]. Es ist also notwendig die Prozedur derart aufzubereiten, dass der Nutzer ihren aktuellen Zustand einfach überblicken kann und somit auch einen Überblick über die nächsten Schritte erhält. Sollte die Prozedur zu komplex sein, bietet sich die Nutzung eines prozedurbasierten Assistenzsystems an (vgl. Kapitel 5.3.1, S. 99).

4.3.2. XML-Darstellung

Zum elektronischen Austausch von Inhalten zwischen verschiedenen Systemen setzt sich zunehmend XML durch. XML ist eine semantisch orientierte Auszeichnungssprache, die in [117] spezifiziert ist. Sie zeichnet sich durch die Möglichkeit aus durch die Definition von XSD-Dateien die Struktur und die Semantik von XML-Dateien formal festzulegen und zu verifizieren. Außerdem können mithilfe von XSLT XML-Dateien von einem Schema in ein anderes Schema transformiert werden [177]. Somit ist es möglich für verschiedene Anwendungsfälle eigene Schemata anzulegen. Beispiele hierfür sind das Modell einer PLT-Stelle [155], das funktionale Anlagenmodell PandIX [55] oder das Abbild der IEC 61131-3 in PLCopen [139].

In der Automatisierungstechnik müssen die Schemata Anforderungen genügen, die in [178] aufgeführt werden. Diese Auflistung von Anforderungen ist in [61] aufgegriffen und ergänzt worden, so dass sie für den Anwendungsfall des Austauschs und der Dokumentation von Prozeduren zutreffend sind. Es muss ein eindeutiger Namensraum zur Identifikation der Elemente vorliegen. Weiter ist ein modulares Konzept erforderlich, das es während des Entwurfsprozesses erlaubt Teilprozeduren auszutauschen. Die so erzeugten Teilprozeduren müssen erweitert und wiederverwendet werden können. Die Erstellung von Prozedurbibliotheken kann auf Basis der XML-Dateien erfolgen. Notwendig ist auch ein Konzept zur Speicherung von Versions- und Erstellerinformationen. Die Möglichkeit des Einfügens nichtstandardisierter Elemente ermöglicht es projektspezifische Informationen in die auszutauschenden Prozeduren aufzunehmen.

Es ist sinnvoll die XML-Darstellung des Referenzmodells auf bereits bestehenden Schemata aufzubauen. Auf diese Weise lassen sich bestehende Konzepte nutzen, die bereits die Anforderungen erfüllen. Zur Auswahl stehen BatchML, Computer Aided Engineering Exchange (CAEX), Web Ontology Language (OWL), OPC UA, PLCopen und SysML, die im Folgenden auf Basis von [61] kurz vorgestellt werden:

- BatchML [119] ist ein Format, mit dem nach der IEC 61512 erstellte Rezepte unab-

hängig von konkreten Anlagen beschrieben werden können. Neben der Modellierung der Prozedur können Versionsinformationen im Rezeptkopf verwaltet werden.

- Mithilfe von CAEX, das in [31] spezifiziert ist, können hierarchisch strukturierte Systeme beschrieben werden. CAEX unterstützt die Definition von Rollen ebenso wie die Modellierung konkreter Systeme. CAEX wird auch durch AutomationML [32] genutzt.

- OWL ist eine Sprache zur Modellierung von Ontologien, d. h. der Darstellung einer Menge von Begriffen, die Beziehungen zueinander haben. Sie ist in [118] spezifiziert. Mithilfe der Sprache können deskriptive Aussagen über die Ontologien getroffen werden.

- OPC UA beschreibt nicht nur eine Möglichkeit der Kommunikation, es enthält auch ein Informationsmodell. OPC UA ist in [20] beschrieben und wird als eines der Basismodelle für Industrie 4.0 angesehen [124].

- Die Programmiersprachen der IEC 61131-3 lassen sich durch PLCopen XML [139] beschreiben. An dieser Stelle ist insbesondere die Möglichkeit interessant, SFC mitsamt ihrer POE und den verwendeten Aktionen auszutauschen.

- Auch zu den Modellierungssprachen der SysML gibt es ein XML-Format [123].

Basierend auf der Analyse von Vor- und Nachteilen der sechs XML-Formate ist in [61] die Wahl auf CAEX als Modellierungsformat für neutrale Prozedurbeschreibungen gefallen. Die Unterstützung hierarchischer Systeme und des Rollenkonzepts sind ebenso wie die Möglichkeit zur Versionierung und die Unterstützung nichtstandardisierter Daten wesentliche Gründe für die Entscheidung. BatchML, PLCopen und SysML sind sehr eng mit den dahinterliegenden Datenmodellen verknüpft, so dass eine Erweiterung dieser Sprachen um die Konzepte des Referenzmodells sehr aufwendig ist. Beispiele hierfür sind die Verwendung von Diensten und die Integration von Menschen als Ausführungseinheit. Zudem wirkt die Kopplung vom Modell mit seiner grafischen Repräsentanz in PLCopen gegen die Unabhängigkeit des Referenzmodells von seiner Visualisierung (vgl. Kapitel 4.3.1, S. 82). Das Informationsmodell von OPC UA ist ein zu mächtiges Werkzeug für die Modellierung des vorgestellten Referenzmodells. Es ist jedoch möglich CAEX-Dateien automatisiert in OPC UA-Modelle zu transformieren [75]. In OWL ist die Verwendung nichtstandardisierter Elemente verboten. Die in [61] entwickelte XML-Darstellung ist für diese Arbeit leicht modifiziert worden und in Anhang D, S. 125, abgedruckt. Die Darstellung wurde sowohl gegen das CAEX-Schema als auch mit dem CAEX-Checker (vgl. [168]) validiert.

4.4. Anforderungen an das Kommunikationssystem und an die Ausführungseinheiten

Auch wenn das verwendete Kommunikationssystem und die Ausführungseinheiten nicht mit dem Referenzmodell modelliert werden, ist es zwingend notwendig, dass sie Anforderungen des Referenzmodells erfüllen.

Das Kommunikationssystem muss grundsätzlich in der Lage sein Nachrichten sowohl zwischen Steuerungssystemen als auch zwischen Steuerungssystem und Ausführungseinheiten auszutauschen. Rein signalbasierte Kommunikationssysteme wie die 4 . . . 20 mA-Technik oder die 0/24 V-Technik können daher nicht eingesetzt werden. Die Übertragung muss bidirektional sein. Des Weiteren muss genügend Bandbreite für die zu übertragenden Nachrichten zur Verfügung stehen. Hierzu ist die Bereitstellung von Quality of Service (QoS)-Informationen[10] notwendig. Diese müssen ebenfalls Daten über die Übertragungsrate beinhalten. Insbesondere sind Informationen über den Determinismus und die Echtzeitfähigkeit des Kommunikationssystems wichtig. Innerhalb des Kommunikationssystems muss ein Adresssystem dafür sorgen, dass die Nachrichtenempfänger ausgewählt werden können. Eine Rückmeldung von Übertragungsfehlern erlaubt der Prozedur, gezielt auf diese Fehler zu reagieren. In einer Industrie 4.0-Umgebung ist zudem der Schutz vor unerlaubten Eingriffen in das Kommunikationssystem wichtig. Der Empfänger muss darauf vertrauen können, dass die Nachrichten tatsächlich von der Einheit gesendet worden sind, die als Absender angegeben ist. Die vorgestellten Anforderungen sind für maschinelle Kommunikationsarten aufgestellt worden, gelten jedoch ebenfalls für die Kommunikation zwischen Mensch und Maschine sowie für die zwischenmenschliche Kommunikation. Jedoch spielt zwischen Menschen auch die Kommunikation über Gesten, Sprache und informalen Text eine entscheidende Rolle. Diese Kommunikationsformen sind den gleichen Anforderungen durch das Referenzmodell unterworfen.

Ausführungseinheiten werden im Referenzmodell über Dienstaufrufe angesprochen. Sie müssen daher in der Lage sein Dienstaufrufe anzunehmen und die entsprechenden Funktionen auszuführen, d. h., sie benötigen eine Eingangsschnittstelle (vgl. z. B. [52]). Zwangsläufig ergibt sich hieraus die Anforderung, dass Ausführungseinheiten aktiv kommunikationsfähig[11] sein müssen. Des Weiteren müssen sie eindeutig adressierbar sein (z. B. über eine Uniform Resource Identifier (URI)[12]). Sie sind also individuell bekannt. Nach [120] müssen Ausführungseinheiten daher die CP-Klassifikation CP33, CP34, CP43 oder CP44 haben. Zur Unterstützung des Entwurfsprozesses der Prozedur müssen Ausführungseinheiten einen Dienstkatalog, auf Englisch Service-Repository, besitzen (vgl. z. B. [77, 143]). Ebenso müssen bei menschlichen Ausführungseinheiten die Fähigkeiten bekannt sein.

Die für das Rollenkonzept notwendigen Informationen werden durch ein Merkmalsystem bereitgestellt. Hierbei ist es sinnvoll auf bestehende Merkmaldatenbanken (beispielsweise eCl@ss oder die IEC 61360 [25]) zurückzugreifen (vgl. [57]). Zusätzlich wird durch eine merkmalbasierte Dienstbeschreibung der Konfigurationsaufwand für ein zweites System eingespart. Ein Dienstinterface ermöglicht den Zugriff auf die Merkmaldaten (vgl. [57, 92]). Das Merkmalsystem und das Dienstsystem benötigen eine Umgebung zur Speicherung der Daten. Dies kann beispielsweise über eine Middleware-Plattform (vgl. z. B. [111]) oder eine Industrie 4.0-kompatible Verwaltungsschale (vgl. [19, 53]) geschehen. Ebenfalls bietet die Kombination aus OPC UA und Field Device Integration (FDI) eine Möglichkeit der dienstbasierten Erkundung einer Ausführungseinheit [19].

[10] Für weitere Informationen zum Thema QoS siehe z. B. [111].

[11] Die Rollen Dienstanbieter und Dienstnutzer sind unabhängig von der Kommunikationsrichtung [128]. Mit anderen Worten, sowohl Ausführungseinheiten als auch Steuerungssysteme müssen Daten senden und empfangen können.

[12] URI sind im RFC 3986 [7] definiert und unterstützen explizit die Adressierung von Menschen.

4.5. Prototypische Implementierung

Für das in Kapitel 4, S. 67, vorgestellte Referenzmodell ist eine prototypische Implementierung in einem maschinellen Steuerungssystem vorgenommen worden. Als Basissystem ist `ACPLT/OV` ausgewählt worden. `ACPLT/OV` ist ein Objektverwaltungssystem für die Prozessautomation, welches in [113] vorgestellt wird. Es wird als Open-Source-Projekt[13] kontinuierlich weiterentwickelt. Ziel dieser Entwicklung ist das Schließen der Lücke zwischen akademischer Forschung und prototypischen Implementierungen. Eine aktuelle Beschreibung der Funktionalität kann [185] entnommen werden.

Neben der Auswahl des Basissystems sind Festlegungen bezüglich des Kommunikationssystems und der Ausführungseinheiten notwendig (vgl. Kapitel 4.4, S. 85). `ACPLT/KS` ist das für diese Implementierung ausgewählte Kommunikationssystem. Der konzeptionelle Aufbau von `ACPLT/KS` ist in [3] dargestellt. `ACPLT/KS` stellt eine Kommunikationsmöglichkeit und ein Adressierungsschema zur Verfügung, die vom Referenzmodell genutzt werden kann. Die prototypische Implementierung berücksichtigt die Themenfelder QoS, Schutz vor unerlaubten Eingriffen und Echtzeitfähigkeit nicht. Ein Schutz vor unerlaubten Eingriffen kann beispielsweise mittels OPC UA erfolgen. Dies ist unter Einbeziehung des Open-Source-Projekts open62541[14] ebenfalls mit `ACPLT/OV` möglich. Verschiedene Bibliotheken, die KS-Schnittstelle `ksapi`, das Funktionsbausteinsystem `fb`, das Merkmalsystem `PropertyManagementSystem`, das Nachrichtensystem `MessageSys` und das Dienstaufrufsystem `ServiceClient` helfen bei der Implementierung des Referenzmodells.

Mit Hilfe von `ACPLT/OV` ist die Implementierung der Klassen des Referenzmodells möglich. In Abbildung 6.1, S. 101, sind diese Klassen mit ihren Attributen vollständig dargestellt. Diese Klassen sind in einer eigenen Bibliothek innerhalb von `ACPLT/OV` implementiert. Des Weiteren enthält die Bibliothek Assoziationen, die die Verknüpfung zwischen Schritten und Transitionen, Schritten und Aktionen usw. ermöglichen. Die Implementierung nutzt die Klasse **SetVar** aus der Bibliothek `ksapi` zum Aufruf einer Prozedur sowie die die Klasse **GetVar** aus derselben Bibliothek für Zustandsabfragen. Dienstaufrufe werden über die Schnittstelle der Bibliothek `ServiceClient` erzeugt und durch die Bibliothek `MessageSys` an den Dienstanbieter geschickt.

Zur Erstellung einer Prozedurbeschreibung können die benötigten Klassen instanziiert und über die Assoziationen miteinander verbunden werden. Neben der Modellierung durch Instanzbildung innerhalb von `ACPLT/OV` ist auch die Modellierung in XML (vgl. Kapitel 4.3.2, S. 84) möglich. Eine XSLT übersetzt die XML-Datei anschließend in ein proprietäres Format [61], das `ACPLT/OV` einlesen kann. Die Implementierung in `ACPLT/OV` bietet den Vorteil, dass die Prozedurbeschreibung im selben System modelliert und ausgeführt wird. Somit können die Vorteile der Erkundbarkeit von Modellen im Zielsystem (vgl. Kapitel 2.6.2, S. 20) ausgenutzt werden.

[13] Siehe `https://github.com/acplt/rte`, Stand 27.01.2016.
[14] Siehe `https://github.com/open62541`, Stand 27.01.2016.

5. Anwendung des Referenzmodells

In diesem Kapitel wird zunächst aufbauend auf der Erläuterung in [130] erklärt, auf welche Weise eine Prozedur mit Hilfe des in Kapitel 4, S. 67, beschriebenen Referenzmodells erstellt werden kann. Anschließend zeigen zwei Beispiele die Anwendung des Referenzmodells zur Modellierung konkreter Prozeduren auf. Diese Beispiele sind eine einfache Pumpenansteuerung und eine Wartungsprozedur für ein Ventil.

5.1. Entwurfsprozess einer Prozedur

Der Entwurf von Prozeduren ist in den Erstellungsprozess der Automatisierungslösung eingegliedert. An diesem Erstellungsprozess sind verschiedene Kontraktoren beteiligt [57]. In [155] sind beispielsweise die fünf Gewerke verfahrenstechnische Planung, Geräteauswahl, Elektroplanung, PLT-Engineering und PLS-Engineering mit Zugriff auf die Konfiguration einer PLT-Stelle genannt. Wünschenswert ist ein umfassendes Modell des Entwurfsprozesses, das alle beteiligten Gewerke zusammenführt. Ein solches Weltmodell ist aber in der Praxis nicht umsetzbar. Stattdessen wird bisher die Entwicklung kleinerer Datenmodelle, die nur einen Teilbereich des Entwurfsprozesses abdecken, als zielführend angesehen [39]. Das in dieser Arbeit vorgestellte Referenzmodell stellt einen solchen Beitrag dar. Es soll helfen, eine automatisierte Übertragung der Prozeduren zwischen den Gewerken zu ermöglichen. Es existiert jedoch kein einheitliches Vorgehen zur Erstellung von Prozeduren. Mit anderen Worten heißt dies, dass die konzeptionelle Vernetzung (vgl. Abbildung 3.1, S. 25) spezifisch für jeden Einzelfall ist. Dennoch ist es möglich einen Einsatz des Referenzmodells als Hilfe zur Modellierung von Prozeduren darzustellen. Zwar ist dieses spezifische Beispiel nicht auf alle denkbaren Fälle eins zu eins übertragbar, es verdeutlicht aber den prinzipiellen Vorgang, wie mit Hilfe des Referenzmodells Prozeduren modelliert werden können.

Die Komplexität der zu planenden Prozedur ist maßgeblich für den Planungsvorgang. So kann beispielsweise eine einzelne Person eine Motorsteuerung entwerfen. Durch die geringe Anzahl an Schritten muss sich die einzelne Person keine Gedanken über Hierarchieebenen zur Gliederung machen. Eine Prozedur für den Anfahrvorgang einer chemischen Großanlage hingegen muss zunächst sowohl bezüglich einer hierarchischen Steuerungsstruktur als auch hinsichtlich der personellen Ausführung der Planungsschritte gegliedert werden. Die Erstellung der hierarchischen Steuerungsstruktur ist auf zwei Arten möglich, mit dem Top-Down-Ansatz oder dem Bottom-Up-Ansatz.

Beide Ansätze basieren auf einem hierarchischen Strukturierungsmodell der Ausführungseinheiten. Beispiele hierfür sind die Anlagenhierarchie einer Batchanlage nach IEC 61512 [35], die Anlagenhierarchie einer Fertigungsanlage [148] oder die Abteilungsstruktur in einem Unternehmen, die in Abbildung 5.1 dargestellt sind.

Startpunkt beim Top-Down-Ansatz ist die höchste Ebene der Ausführungseinheitsstruktur. Dieser höchsten Ebene ist eine Prozedur zugeordnet, die das Zusammenwirken der ein-

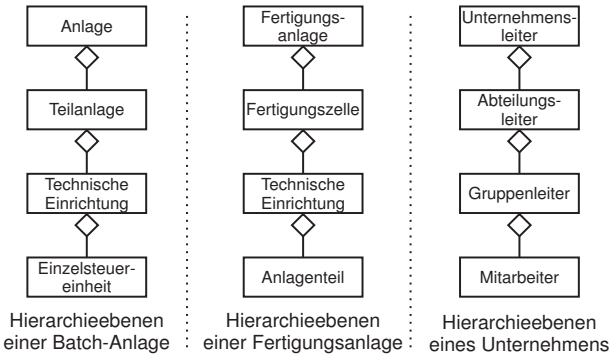

Abbildung 5.1.: Beispiele für hierarchische Strukturen von Ausführungseinheitstypen

zelnen Elemente der zweithöchsten Ebene steuert. Im Referenzmodell können die Elemente der zweithöchsten Ebene Dienste anbieten, die durch die Prozedur orchestriert werden. Dieses Vorgehen wird iterativ angewendet, bis die niedrigste Ebene erreicht ist, die direkten Zugriff auf die Einzelsteuereinheiten hat. Ein Beispiel hierfür ist eine Anfahrprozedur für eine Anlage, bei der zunächst die Reihenfolge des Anfahrens der unterschiedlichen Teilanlagen bestimmt wird. Erst im weiteren Verlauf des Entwurfsprozesses erfolgt die Bestimmung der konkreten Aktoren, die zum Anfahren der Teilanlage benötigt werden.

Der Bottom-Up-Ansatz verwendet die umgekehrte Vorgehensweise. Hier ist der Ausgangspunkt des Prozedurentwurfs das Erzeugen von Prozeduren, welche Elemente der untersten Schicht der Hierarchie der Ausführungseinheiten ansteuern. Diese werden sukzessiv von den Prozeduren zur Steuerung der höheren Ebenen aufgegriffen und über Dienste aufgerufen.

Beide Wege führen zu einer Prozedur, die zur Steuerung des gewünschten Prozesses auf die Ausführungseinheiten einwirken kann. Durch die Verwendung von Diensten ist die Aufteilung der Verantwortung für die Gesamtprozedur an mehrere Prozedurplaner möglich, wie in [76] beschrieben wird. Notwendige Voraussetzung ist eine exakte Definition der Dienstbeschreibungen, welche die Schnittstelle zwischen den verschiedenen Teilprozeduren bilden.

Die entworfene Prozedur muss vom Steuerungssystem interpretiert und ausgeführt werden können, d. h., die Prozedur muss der Syntax und der Semantik (vgl. Kapitel 2.6.1, S. 19) des Zielsystems entsprechen. Ein menschliches Steuerungssystem muss kognitiv dazu in der Lage sein die Abläufe zu verstehen. Diese Herausforderung wird in Kapitel 5.3, S. 97, thematisiert. Zunächst liegt der Fokus jedoch auf maschinellen Steuerungssystemen. Die Betrachtung maschineller Steuerungssysteme zeigt, dass derzeit eine Transformation der Prozedurbeschreibung vom hier vorgestellten Referenzmodell über domänenspezifische Engineering-Modelle in die Implementierungssprachen der Steuerung notwendig ist [152]. Eine solche Umwandlung kann durch Modelltransformationen und modellgetriebene Codegenerierung (vgl. Kapitel 2.6.2, S. 20) weitgehend automatisiert erfolgen. Hierzu sind vier Implementierungsschritte notwendig [130, 152]:

1. Die Prüfung der Prozedurbeschreibung auf Übereinstimmung mit dem Meta-Modell stellt sicher, dass die nachfolgenden Transformationsschritte ohne formalen Fehler ablaufen können.

2. Die allgemeine Prozedurbeschreibung kann (z. B. durch eine XSLT) in eine domänenspezifische Sprache transformiert werden. Ein Beispiel hierfür ist eine Transformation aus der XML-Darstellung in Kapitel 4.3.2, S. 84, nach PLCopen [139].

3. Nach der Prüfung der im Zielsystem vorhandenen Systemstrukturen erfolgt die händische oder teilautomatische Zuordnung der Ausführungseinheiten zu den Signalen im Zielsystem.

4. Eine modellgetriebene Codegenerierung erzeugt den Programmcode für das Zielsystem.

Dieses Vorgehen ist ohne Modifikation der bestehenden maschinellen Steuerungssysteme möglich. Prozedurplaner können folglich den Umgang mit dem vorgestellten Referenzmodell ohne Investitionen in neue Steuerungssysteme trainieren. Dies erhöht die Akzeptanz des Referenzmodells bei den Anwendern. Zudem vereinfacht sich der Aufwand zur Verifikation und zur Validierung des Steuerungscodes (vgl. z. B. [62, 96]). Statt des systemspezifischen Programmcodes muss lediglich die allgemeine Prozedurbeschreibung und das Mapping der Dienste auf die Signale geprüft werden. Eine Hilfe beim Mapping der Dienste auf die Signale im bestehenden Steuerungssystem ist ein Modell der PLT-Stellen nach der NE 150 [129], wie Abbildung 5.2 zeigt.

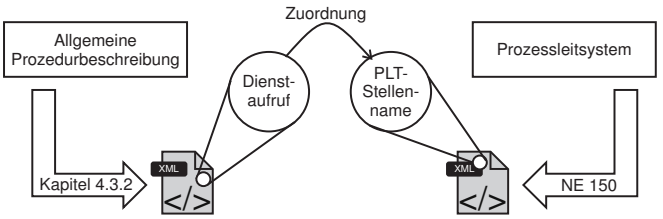

Abbildung 5.2.: Nutzung der NE 150 zur Zuordnung von Diensten und Signalen

Das Meta-Modell der PLT-Stelle enthält die Beziehung zwischen PLT-Stellenname und dem zugeordneten Signal. Die PLT-Stellen können in eine XML-Datei exportiert werden [155]. Anschließend ist die Zuordnung zwischen Dienstaufrufen und Signalen auf dieser Grundlage möglich und eine PLCopen-kompatible Datei kann erstellt und in das PLS geladen werden.

5.1.1. Flexible Strukturen

Wie bereits in Kapitel 2.6.2, S. 20, erläutert, ist die Verwendung von Modellen im Zielsystem der modellgetriebenen Codegenerierung vorzuziehen. Eine flexible Anpassung der

Prozeduren durch Anlagenfahrer, die beispielsweise im Zukunftsprojekt Industrie 4.0 [91] vorgesehen ist, macht bei ausschließlicher Verwendung der modellgetriebenen Codegenerierung die Ausführung aller vier Implementierungsschritte notwendig. Insbesondere ist die Zuordnung der Ausführungseinheiten zu den Signalen im Zielsystem zu komplex, zu zeitaufwendig und damit zu teuer für den Anlagenfahrer, so dass ein Prozedurplaner hinzugezogen werden sollte.

Zusammenfassend ist die Implementierung eines Dienstsystems in die Zielsysteme unabdingbar [130]. Dies beginnt auf der Ebene 1 der Automatisierungspyramide (vgl. Kapitel 2.4, S. 14). Hier findet die „Übersetzung“ zwischen dienstbasierter und signalbasierter Kommunikation statt, die in Abbildung 5.3 demonstriert ist.

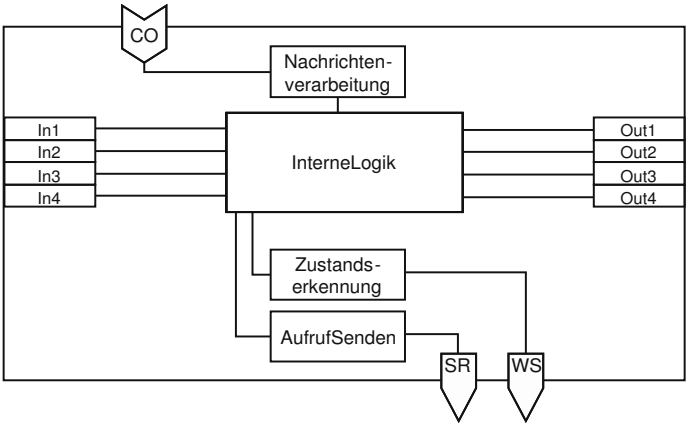

Abbildung 5.3.: Erweiterung einer Einzelsteuerung um ein Dienstinterface (basierend auf [51])

Dazu ist eine Erweiterung der Steuerungssysteme um vier IEC 61131-3 -Blöcke erforderlich, von denen drei in [51] erwähnt werden:

- Der Funktionsblocktyp **CO** empfängt die Dienstaufrufe der Prozedur als Nachricht.
- Der Funktionsblocktyp **Nachrichtenverarbeitung** interpretiert die eingehenden Nachrichten und beeinflusst entsprechend die **InterneLogik** der Einzelsteuerung.
- Der Funktionsblocktyp **AufrufSenden** steuert in Abhängigkeit des Zustands der internen Logik das Absenden von Dienstaufrufen über den Ausgang **SR**.

Neben den in [51] erwähnten Blöcken ist ein Funktionsbausteintyp **Zustandserkennung** notwendig. Während der Block **AufrufSenden** zu diskreten Zeitpunkten Dienstaufrufe absendet, ermittelt der Block **Zustandserkennung** fortlaufend den aktuellen Status der Einzelsteuerungseinheit aus den Eingangssignalen. Der ermittelte Zustand **WS** kann in den Transitionsbedingungen der aufrufenden Prozedur abgefragt werden.

Aufbauend auf die um eine Dienstschnittstelle erweiterten Einzelsteuerungen in der Ebene 1 können nun Prozeduren in den höheren Ebenen der Automatisierungspyramide auf die

bereitgestellten Dienste zugreifen. Die Dienstaufrufe lassen sich zur Laufzeit der Prozedur umkonfigurieren. Somit ist eine flexible Anpassung der Prozedur durch den Anlagenfahrer an aktuelle Gegebenheiten möglich. Denkbar ist auch die Implementierung einer Datenbank für Teilprozeduren, die durch die Anlagenfahrer zur Laufzeit konfiguriert und in einer an die Situation angepassten Reihenfolge gestartet werden. Die erforderlichen Vorbedingungen können in der Transition nach dem Initialschritt der Prozedur hinterlegt sein. Der Anlagenfahrer fungiert so als menschliches Steuerungssystem (vgl. Kapitel 5.3, S. 97). Ein Konzept zur Suche der passenden Dienste kann [109] entnommen werden.

Dieses Vorgehen hat zwei positive Nebeneffekte. Erstens können modulare Anlagen und Package Units einfach in eine Anlage integriert werden und zweitens wird eine Verlagerung von Intelligenz auf Feldgeräte gefördert. Die Funktionssicherheit der Anlage wird nicht beeinträchtigt:

- Die Steuerung modularer Anlagen (vgl. z. B. [133, 135]) kann analog zu den Einzelsteuereinheiten um die vier IEC 61131-3 -Blöcke ergänzt werden. Aus diese Weise wird eine Abstraktion von den herstellerspezifischen Ausführungen der Steuerung geschaffen.

- Die Implementierung der Einzelsteuerung auf Feldgeräten führt zu einer besseren Verteilung der Last im gesamten Automatisierungssystem [57]. Bezogen auf das Referenzmodell lässt sich eine bessere Wiederverwendbarkeit der Prozeduren sicherstellen, wenn statt Messgrößen Zustandsinformationen mit Semantik (z. B. „Behälter ist halb voll“ statt $L1.PV = 25\,\mathrm{cm}$) übertragen werden. Dies kann durch die Definition von Merkmalen mit einer Beschreibung erfolgen. Die konkreten Messwerte werden nicht benötigt, da keine kontinuierliche Regelung in einer Prozedur durchgeführt wird.

- In [51] werden drei Ebenen der funktionalen Integrität eingeführt, Addon-Funktionen, Basis-Funktionen und sicherheitsrelevante (Safety-) Funktionen. Diese sind in Abbildung 5.4 dargestellt.

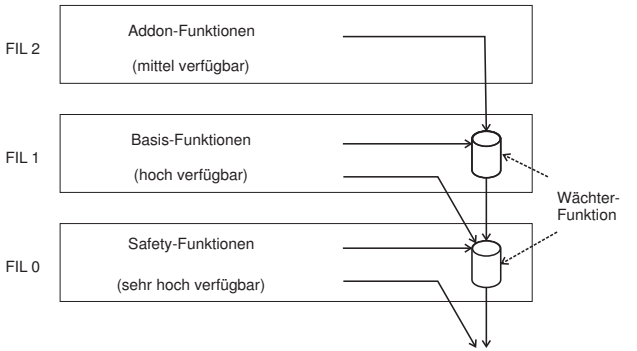

Abbildung 5.4.: Ebenen der funktionalen Integrität (nach [51])

Die sicherheitsrelevanten Funktionen (z. B. Verriegelungen) werden in den Einzelsteuerungen wie gewohnt implementiert. Somit wird auf eine bewährte Technik zurückgegriffen. Das hier vorgestellte Referenzmodell kann für Funktionen in den Ebenen FIL 1 und FIL 2 genutzt werden. Das ServiceInterface der Ebene FIL 0 kann gleichzeitig als Wächter-Funktion alle Zugriffe abblocken, welche die funktionale Sicherheit bedrohen.

5.2. Steuerungsprozedur einer Pumpe

Das erste Beispiel zeigt anhand einer einfachen Steuerungsprozedur einer Pumpe die prinzipielle Anwendung des Referenzmodells. Die Pumpe befindet sich in einer LKW-Abfüllung, bei der ein LKW über die Pumpe und ein Ventil aus einem Tank befüllt wird (vgl. Abbildung 5.5).

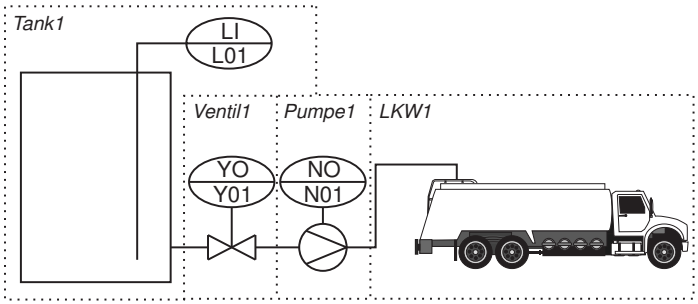

Abbildung 5.5.: Schematische Darstellung einer LKW-Abfüllstation

In Abbildung 5.5 sind vier verschiedene Teilsysteme zu erkennen, *Tank1, Ventil1, Pumpe1* und *LKW1*. Im Beispiel wird die Steuerung des Teilsystems *Pumpe1* betrachtet[1]. *Pumpe1* ist eine Ausführungseinheit, die die PLT-Stelle N01 beinhaltet. Die Einzelsteuerung von N01 ist analog zu Abbildung 5.3 erweitert worden und in Abbildung 5.6 abgebildet.

Gegenüber den konventionellen Funktionsbausteinen sind die Bausteine **StringCompare** und **Switch** ergänzt worden. Der Baustein **StringCompare** liefert eine boolesche Eins als Ausgang, wenn die beiden Texte des Eingangs übereinstimmen. Der Baustein **Switch** gibt den ersten Text als Ausgang weiter, wenn die boolesche Eingangsvariable Null ist. Andernfalls wird der zweite Text ausgegeben. Neben der Einzelsteuerung von N01 müssen auch die andern Teilsysteme in der Steuerungsprozedur von *Pumpe1* berücksichtigt werden. Die weiteren Teilsysteme bieten die folgenden Zustandsmeldungen an:

- *Tank1*: TankVoll, TankLeer,
- *Ventil1*: Auf, Zu und

[1]Das Öffnen und Schließen des Ventils im Normalbetrieb wird der Übersichtlichkeit halber außen vor gelassen werden.

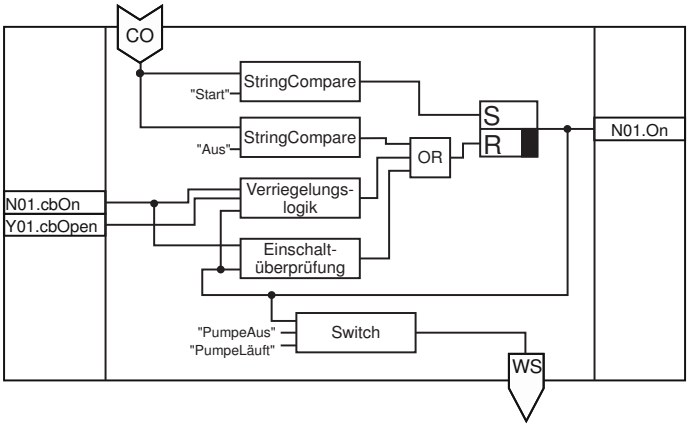

Abbildung 5.6.: Einzelsteuerung der Pumpe N01

- *LKW1*: LKWPositioniert, LKWVoll.

Eine überlagerte Steuerung (vgl. Kapitel 2.4, S. 14) kann die Befehle „Start“ und „Ausschalten“ an die Steuerung der Abfülleinheit übergeben. Mit diesen Informationen ist es möglich eine Abfüllprozedur wie folgt textuell zu formulieren: Nach dem Starten der Abfüllanlage wird die Pumpe gestartet, sobald der LKW positioniert ist. Wenn der LKW befüllt ist, stoppt sie wieder. Als mögliche Fehlerfälle während des Laufens der Pumpe sind das Leerlaufen des Tanks, das Schließen des Ventils oder das Entfernen des LKWs bekannt.

Dieser informell beschriebene Ablauf lässt sich mit Hilfe des in Kapitel 4, S. 67, vorgestellten Referenzmodells formalisiert darstellen. In Abbildung 5.7 ist ein Instanzmodell dargestellt, das diesen Ablauf modelliert.

Es sind insgesamt acht Instanzen des Aufbaumodells angelegt worden, vier Transitionen (***S1S2, S2S1, S2S4*** und ***S1S3***), ein Anfangsschritt (***S1***), ein elementarer Schritt (***S2***) und zwei Endschritte (***S3*** und ***S4***). Mit den Schritten sind insgesamt vier Instanzen (***A1*** bis ***A4***) des Typs Dienstaufruf über Assoziationen verbunden. Die Instanzen der Dienstaufrufe sind wegen der Übersichtlichkeit in Abbildung 5.8 abgebildet.

Jede Transition ist mit genau einer Bedingung (***C1*** bis ***C4***) verbunden. Die Transitionsbedingungen, die in Abbildung 5.7 verwendet werden, sind in Abbildung 5.9 dargestellt.

Eine Transitionsbedingung besteht aus ein oder mehreren logischen Termen. Die Bedingungen ***C1*** und ***C2*** ermitteln jeweils den Zustand des Teilsystems *LKW1* (***SC1***, ***SC2*** und ***SC3***). Die Bedingung ***C3*** prüft, ob die Pumpensteuerung den Befehl Ausschalten erhalten hat (***RR1***). ***C4*** ist eine zusammengesetzte Bedingung, die aus drei mit Oder verknüpften Zustandsabfragen (***SC4*** bis ***SC6***) besteht.

Die Steuerung kann von außen über einen Befehl Start eingeschaltet werden. Dieser aktiviert den Schritt ***S1*** und der Dienstaufruf ***A1*** wird ausgeführt. Dies stellt sicher, dass die Pumpe abgeschaltet ist. Sobald ein nicht voller LKW *LKW1* in der Abfüllanlage positioniert ist, feuert die Transition ***S1S2***, die den Schritt ***S2*** aktiviert. Nun läuft die Pumpe

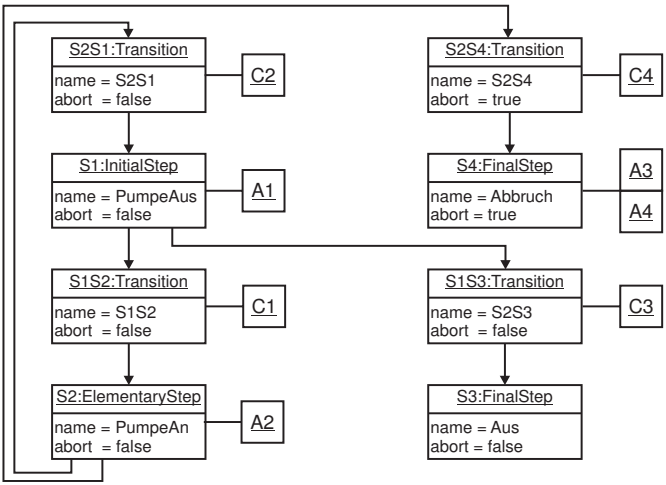

Abbildung 5.7.: Steuerungsprozedur für die Pumpe der LKW-Abfüllstation

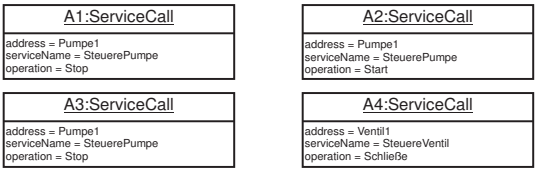

Abbildung 5.8.: Verwendete Aktionen in Abbildung 5.7

so lange, bis der LKW *LKW1* voll ist. Dies führt zum Feuern der Transition ***S2S1*** und die Pumpe schaltet ab (Schritt ***S1***). Eine erneute Aktivierung der Pumpe kann erst erfolgen, wenn erneut ein nicht voller LKW positioniert wird. Die Abfüllanlage kann mit dem Befehl Ausschalten über ***S1S3*** nur ausgeschaltet werden, wenn kein Abfüllvorgang stattfindet. Ist die Pumpe eingeschaltet, führt ein Leerlaufen des Tanks *Tank1*, ein Entfernen des LKWs *LKW1* oder ein Schließen des Ventils *Ventil1* zum Auslösen der Abbruchtransition ***S2S4***.

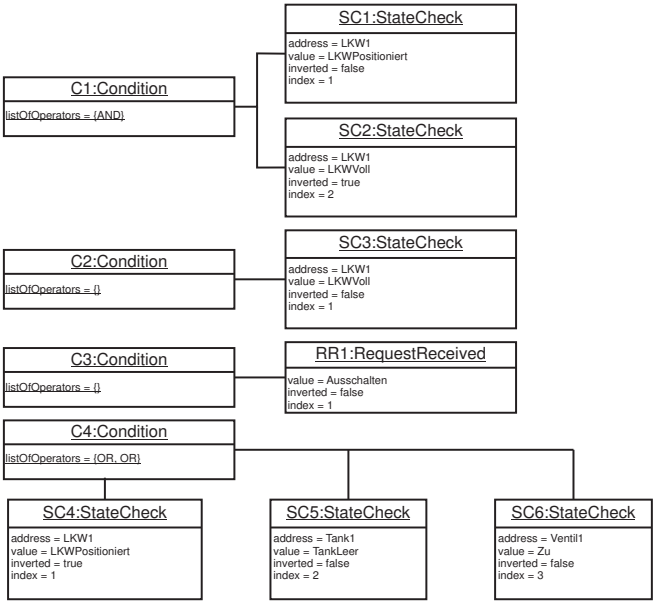

Abbildung 5.9.: Verwendete Transitionsbedingungen und logische Terme in Abbildung 5.7

5.2.1. Verwendung von Rollen

In Kapitel 4.2.4, S. 76, ist die Verwendung von Rollen in der Prozedurbeschreibung als Möglichkeit der Erhöhung der Flexibiliät bereits beschrieben. An dieser Stelle verdeutlicht die Anwendung auf das beschriebene Beispiel den Vorteil. In Abbildung 5.9 ist erkennbar, dass sich die Zustandsüberprüfungen auf einen konkreten LKW, nämlich *LKW1*, beziehen. Falls ein anderer LKW befüllt werden soll, muss die Prozedur bearbeitet werden.

Der erste Schritt zur Flexibilisierung der Prozedur besteht in der Verwendung von Parametern. Statt der Verwendung der konkreten Bezeichnung LKW1 lässt sich ein Parameter im Ausführungsrahmen definieren, der als Adresse in den Zustandsabfragen ***SC1***, ***SC2*** und ***SC3*** verwendet werden kann. Bei jedem eintreffenden LKW wird der Wert des Parameters entsprechend geändert.

Dies führt zu einer Flexibilität der Prozedur, erhöht aber den manuellen Aufwand. So muss beispielsweise für jeden LKW überprüft werden, ob er die benötigten Zustände für die Prozedur bereithält. Mit Hilfe von Rollen lässt sich die Zuordnung automatisieren. In der Zustandsabfrage wird eine Rolle referenziert, die die Rückmeldung der Zustände LKWPositioniert und LKWVoll als Anforderung beinhaltet. Abbildung 5.10 zeigt beispielhaft die Verwendung von Rollen in der Zustandsabfrage ***SC3***. Da das Merkmal ***P1*** die Anforderung ***R1*** erfüllt, kann die Ausführungseinheit ***LKWxy*** die Rolle ***LKW*** ausfüllen und somit durch ***SC3*** angesprochen werden.

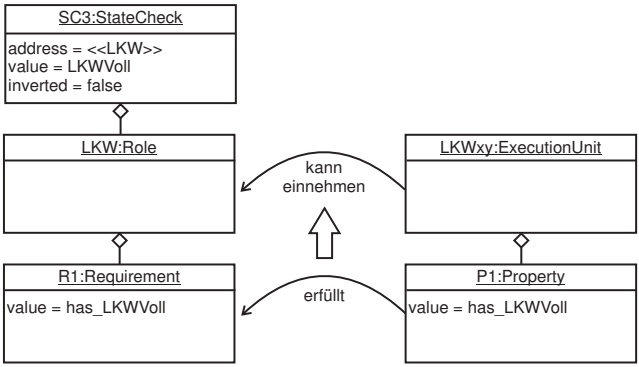

Abbildung 5.10.: Beispiel für die Zuordnung von Ausführungseinheiten zu einer Rolle in einer Prozedurbeschreibung

5.3. Integration von Menschen in die Prozedurausführung

Wie bereits in Abbildung 2.4, S. 13, erläutert, können Menschen sowohl als Steuerungssystem als auch als Ausführungseinheit an einer Prozedurausführung beteiligt sein. Ein Beispiel hierfür ist die Reparatur eines Regelventils. Bei diesem Ventil sei die Membran des Stellorgans gerissen, so dass der notwendige Druck zum Einstellen der Ventilposition nicht mehr aufgebracht werden kann. Dies führe dazu, dass der Prozess zwar weiter

betrieben werden kann, er sich aber nicht mehr im Produktionsoptimum befindet. In Abbildung 5.11 ist ein UML-Sequenzdiagramm abgebildet, das die notwendigen Schritte zur Reparatur in ihrer zeitlichen Abfolge darstellt. Aus Gründen der Übersichtlichkeit sind die zugehörigen Klassenmodelle der Prozeduren in Anhang B, S. 114, abgebildet.

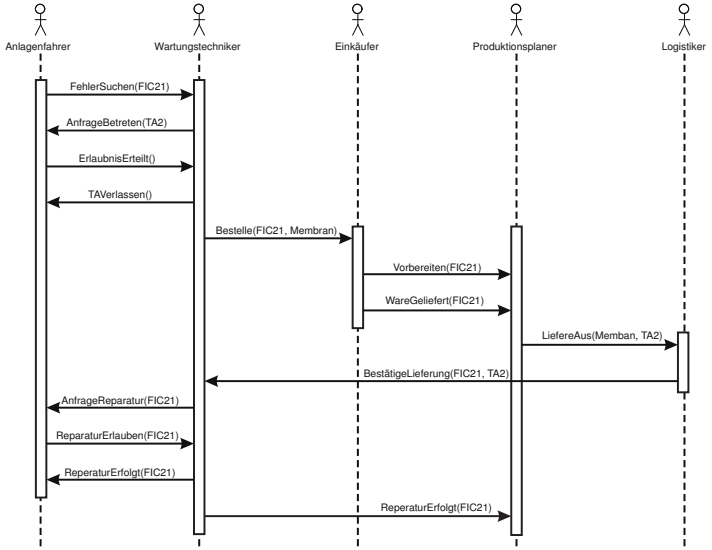

Abbildung 5.11.: Sequenzdiagramm zur Darstellung der zeitlichen Abfolge von Dienstaufrufen während einer Ventilwartung

Auslöser des Wartungsvorgangs ist die Feststellung eines zu niedrigen Durchflusses an der Durchflussregelung ***FIC21***. Der Anlagenfahrer versucht zunächst z. B. mit Hilfe anderer Messgrößen, einen defekten Sensor als Fehlerursache auszuschließen. Ist dies nicht der Fall, veranlasst er, dass der Wartungstechniker das Ventil überprüft. Mit anderen Worten, er sendet über sein HMI den Befehl `Start` sowie den Parameter `FIC21` an die Prozedur ÜberprüfeRegelventil des Wartungstechnikers. Der Wartungstechniker wählt die für die Diagnose notwendigen Werkzeuge aus und meldet sich beim Anlagenfahrer zum Betreten der Teilanlage an, sobald er vor Ort ist. Der Anlagenfahrer prüft, ob im aktuellen Betriebszustand der Anlage ein Betreten gefahrlos möglich ist und erteilt dann die Erlaubnis. Der Wartungstechniker sucht anschließend die Fehlerursache und bestellt nach Verlassen der Anlage das notwendige Ersatzteil. Gleichzeitig informiert er den Produktionsplaner über die Wartung, die dieser in seinen Produktionsplan einplanen muss. Nach der Bestellung des Ersatzteils unterrichtet der Einkäufer den Produktionsplaner über den geplanten Liefertermin zur groben Einplanung der Wartung. Sobald die Lieferung tatsächlich eingetroffen ist, informiert der Produktionsplaner den Logistiker und den Wartungstechniker über den genauen Zeitpunkt der Wartungsmaßnahme. Vor der Reparatur gibt der Anlagenfahrer die Anlage zur Reparatur frei.

Anhand des Beispiels wird deutlich, welche Entscheidungen die fünf Akteure Anlagenfahrer, Wartungstechniker, Einkäufer, Produktionsplaner und Logistiker zu treffen haben. Die Funktionen der Akteure sind auf die drei unteren Ebenen der Automatisierungspyramide verteilt. Jeder Akteur besitzt seine eigenen Prozeduren, die in Anhang B, S. 114, dargestellt sind. Diese sind, wie durch das Sequenzdiagramm deutlich wird, durch Querabhängigkeiten miteinander verbunden.

Der Produktionsplaner beispielsweise nutzt eine Prozedur zur Modifikation seines Produktionsplans, der wiederum eine Prozedur ist. Der Anlagenfahrer nutzt die Rezepte zur Steuerung des Produktionsprozess. Der Wartungstechniker kann Prozeduren als Anleitung des Reparaturprozesses verwenden. Des Weiteren stehen die Prozeduren, die die Arbeitsabläufe der Akteure steuern, in einem engen Zusammenhang. Es muss demnach zum einen eine Schnittstelle zwischen den Systemen existieren, zum anderen müssen die Steuerungssysteme eine einheitliche Syntax und Semantik unterstützen. Auf diese Weise ist eine engere Koordination des komplexen Reparaturprozesses möglich. Das in dieser Arbeit vorgestellte Referenzmodell ermöglicht die Bildung einer Brücke zwischen den verschiedenen Prozeduren. In der täglichen Anwendung muss der Fokus auf der Gestaltung des HMI liegen (vgl. [169]). Die Schnittstelle zwischen Mensch und Steuerung kann hierbei durch ein Assistenzsystem erfolgen.

5.3.1. Assistenzsysteme

Zur Reduktion der Komplexität der zu erledigenden Vorgänge und zur Verdeutlichung der nächsten Steuerungsschritte können Assistenzsysteme eingesetzt werden [170]. Dem Assistenzsystem stehen drei Informationsquellen zur Verfügung, die Vorgabe des Prozessziels, die Zustandsinformationen der Anlage und die Informationen, die vom Operator eingegeben worden sind [70]. Des Weiteren sind Modelle und Methoden hinterlegt, aus denen Entscheidungshilfen abgeleitet werden können [153]. Mit Hilfe dieser Informationen schlägt ein Assistenzsystem die Brücke zwischen Wünschen, Zielen, Fähigkeiten und Wissen eines Menschens und den Funktionen eines interaktiven maschinellen Systems [187]. Assistenzsysteme können jedoch nicht nur während des operativen Betriebs einer Anlage unterstützend tätig sein, auch der Entwurfsprozess kann assistiert werden [85].

Eine Integration des hier vorgestellten Referenzmodells in Assistenzsysteme ist möglich, wie in [153] gezeigt ist. Sobald das Assistenzsystem eine unterstützungsbedürftige Situation erkennt, sucht es in einer Prozedurdatenbank nach einer aufgezeichneten Prozedur. Diese Dokumentation einer Prozedurausführung gibt dem Anlagenfahrer Hinweise, wie ein ähnliches, schon einmal aufgetretenes Problem gelöst worden ist. Der Anlagenfahrer interpretiert diese Information und versucht die richtigen Schlüsse zu ziehen. Gefüllt weird solch eine Prozedurdatenbank entweder manuell oder mittels der Methode des Fallbasierten Schließens [153].

Im Beispiel der Ventilreparatur sind verschiedene Einsatzmöglichkeiten gegeben. So kann ein Assistenzsystem dem Anlagenfahrer helfen, die Ursache für die Prozessstörung zu ermitteln. Dem Wartungstechniker können die Reparaturschritte interaktiv auf einem industriellen Tablet visualisiert werden. Der Produktionsplaner erhält z. B. Vorschläge für eine optimale Anordnung der verschiedenen Rezepte.

6. Zusammenfassung und Diskussion

6.1. Zusammenfassung

Diese Arbeit befasst sich mit einem Referenzmodell zur Beschreibung allgemeiner Prozeduren. In Kapitel 4, S. 67, ist dieses Referenzmodell basierend auf den Elementen einer Prozedurbeschreibungssprache (vgl. Kapitel 3.1, S. 24) beschrieben. Zum Aufbaumodell gehören die Elemente Ausführungsrahmen (**ExecutionFrame**), Transition (**Transition**) und Schritt (**Step**), wobei vom Schritt die Klassen Elementarschritt (**ElementaryStep**), Anfangsschritt (**InitialStep**) und Endschritt (**FinalStep**) abgeleitet sind. Im Hierarchie- und Vernetzungsmodell sind Verknüpfungsregeln zur Erzeugung von linearen Ketten sowie zur Alternativ- und Parallelverzweigung aufgestellt worden. Zudem ermöglicht die Klasse Makroschritt (**MacroStep**) die Bildung von Hierarchien. Das Aktions- und Aktivitätenmodell legt die Interaktionsmöglichkeiten mit der Umgebung fest. Zum einen werden die Klassen Dienstaufruf (**ServiceCall**) und Prozeduraufruf (**ProcedureCall**) als Aktionen definiert, zum anderen kann sich die Transitionsbedingung aus logischen Termen zusammensetzen. Das Abstraktions- und Zuordnungsmodell beschreibt die Möglichkeiten, den Entwurfsprozess der Prozedur zu vereinfachen, während das Ausführungssteuerungsmodell die Ausführung der Prozedurbeschreibung determiniert.

Bei der Vorstellung des Referenzmodells in Kapitel 4, S. 67, sind der Übersichtlichkeit wegen lediglich Ausschnitte aus der UML-Darstellung des Referenzmodells dargestellt worden. In Abbildung 6.1 ist das vollständige Klassendiagramm abgebildet. Neben der Modellierung des eigentlichen Referenzmodells (graues Rechteck) sind auch das Zusammenwirken des Referenzmodells mit einem Visualisierungsmodell (oberhalb des grauen Kastens) und die Verwendung von Rollen, Merkmalen und Ausführungseinheiten (unterhalb des grauen Kastens) abgebildet.

Mithilfe dieses Referenzmodells können verschiedene Typen von Prozeduren abgebildet werden. In Kapitel 5, S. 88, ist gezeigt worden, dass es möglich ist, sowohl die Steuerprozedur einer Pumpe als auch die Wartungsprozedur eines Ventils zu beschreiben. Diese beiden Beispiele zeigen aufgrund ihrer stark unterschiedlichen Komplexität die Allgemeingültigkeit des Modells. Dadurch wird deutlich, dass das Modell auf allen Ebenen der Automatisierungspyramide eingesetzt werden kann. Ebenso wird die Integration von Menschen sowohl auf Steuerungs- als auch auf Ausführungsseite unterstützt.

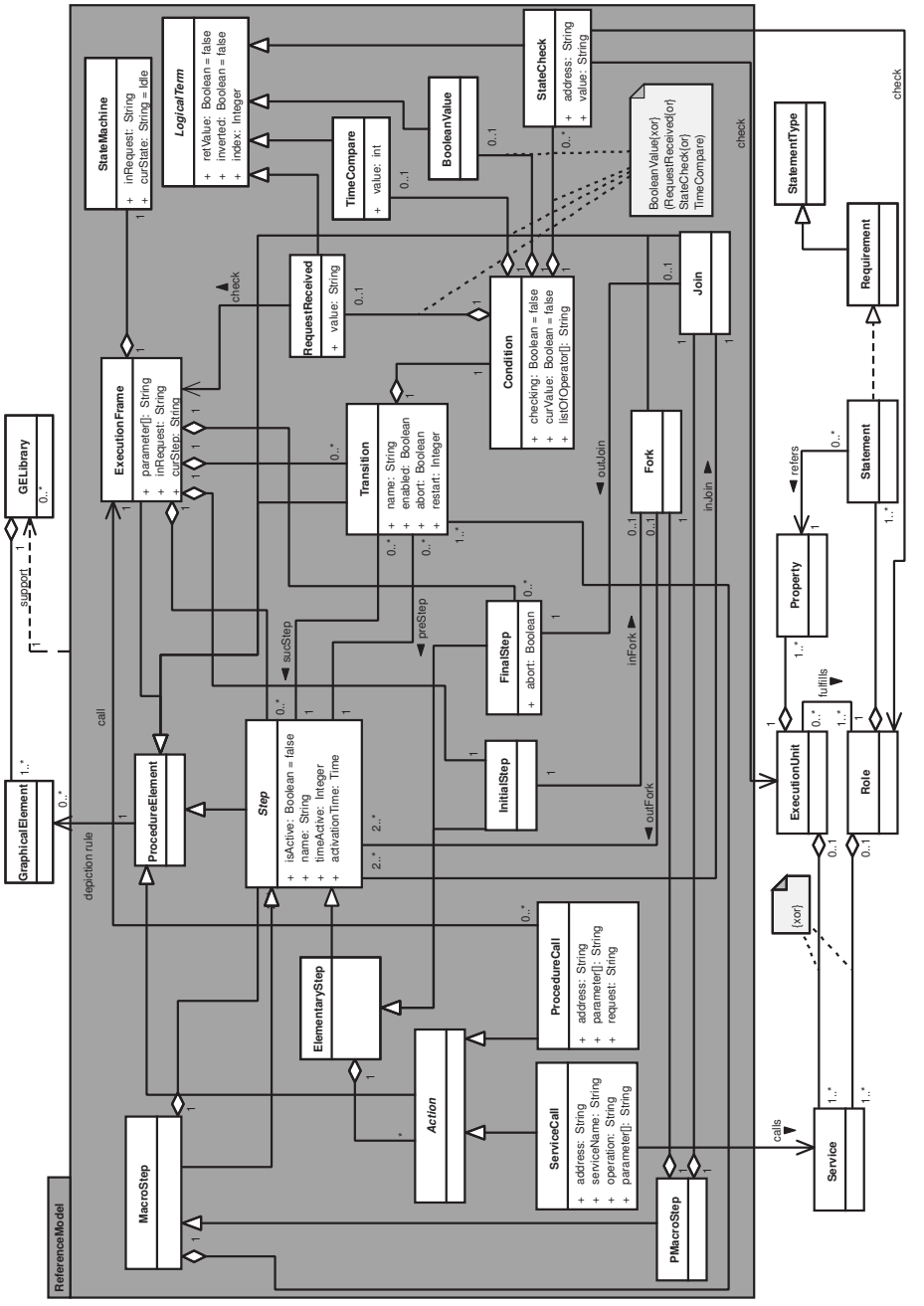

Abbildung 6.1.: Klassendiagramm

6.2. Diskussion

Basis für die Erstellung des Referenzmodells ist die Analyse bestehender Prozedurbeschreibungssprachen. Durch die Analyse ist der gemeinsame Kern ermittelt worden, der den Prozedurbeschreibungssprachen zugrunde liegt. Das Referenzmodell deckt diesen gemeinsamen Kern ab. Mithilfe des Referenzmodells können lineare Prozeduren bestehend aus Schritten und Transitionen modelliert werden. Des Weiteren werden auch Makroschritte, Alternativverzweigungen und nebenläufige Prozeduren unterstützt. In Anhang C, S. 119, sind einige Beispiele dargestellt, bei denen die Vermutung naheliegt, dass sie durch das Referenzmodell nicht abgedeckt werden. Die Beispiele zeigen jedoch, dass dies nicht der Fall ist und geben gleichzeitig eine Empfehlung für die Modellierung ab.

Das Modell besticht durch seine Einfachheit, d.h., ein Vorteil besteht darin, dass durch wenige Modellelemente Abläufe beschrieben werden können. Die Einfachheit des Referenzmodells basiert hauptsächlich auf der konsequenten Anwendung der Prinzipien der Kybernetik. In der Prozedurbeschreibung werden nur die Dienstaufrufe und Zustandsabfragen modelliert, d. h., die Ausführungseinheiten verbergen die Art und Weise, wie sie die aufgerufenen Funktionen ausführen, vor der Prozedur. Auf diese Weise kann die Komplexität von der Prozedur auf die Ausführungseinheiten verlagert werden. Wichtig ist an dieser Stelle der Hinweis, dass eine Ausführungseinheit wiederum eine Steuerung besitzen kann, die die Unter-Ausführungseinheiten kontrolliert. Diese strikte Trennung zwischen Aufruf einer Funktion und ihrer Ausführung vereinfacht die Beschreibung einer Prozedur deutlich.

Die Verwendung von Dienstaufrufen vereinheitlicht den Aufruf verschiedener Funktionstypen. Ein Dienstaufruf kann beispielsweise das Setzen einer Variablen, das Starten eines Batch-Prozesses oder eine Anordnung an einen Mitarbeiter sein. Die verschiedenen Typen können ohne Weiteres gemeinsam in einer Prozedur verwendet werden. Die Verwendung dieser einheitlichen Schnittstelle zwischen Steuerungssystem und Ausführungseinheiten fördert eine bessere Wartung und eine Erhöhung der Erweiterbarkeit der Prozedur. Des Weiteren ist die Übermittlung von formalisierten und nicht-formalisierten Aufrufen an unterschiedliche Empfängertypen (vgl. Abbildung 2.4, S. 13) in derselben Weise möglich. Die Kommunikation durch Dienste findet immer zwischen genau zwei Kommunikationspartnern statt, dem Dienstaufrufer (hier die Prozedur im Steuerungssystem) und dem Dienstanbieter (hier die Ausführungseinheit). Dies minimiert die Belastung der im Kommunikationssystem eingebundenen Teilnehmer, da jeder Teilnehmer nur die Nachrichten verarbeiten muss, die auch wirklich an ihn gerichtet sind. Das Konzept der Dienstaufrufe ermöglicht zudem die Einbindung externer Dienstanbieter. Da die Art und Weise, wie der Dienst ausgeführt wird, nicht an den Dienstaufrufer weitergegeben wird, ist ein Know-How-Schutz für den Dienstanbieter gewährleistet.

Dienstaufrufe sind neben dem Rollenkonzept die grundlegende Basis für flexible Prozeduren. Durch die Dienstaufrufe werden feste, bereits im Planungsprozess der Prozedur definierte Signalverbindungen vermieden. Dies steht nicht im Widerspruch zu der bestehenden konventionellen Verdrahtung von Sensoren und Aktoren in bestehenden chemischen Anlagen. In Kapitel 5.1.1, S. 90, ist gezeigt, wie eine konventionelle Einzelsteuerung in ein dienstbasiertes System eingebunden werden kann. Es ist möglich Dienstaufrufe während des Entwurfsprozesses abstrakt mit Bezug zu einer Rolle zu definieren. Die Zuordnung einer konkreten Ausführungseinheit erfolgt entweder im weiteren Verlauf des Entwurfsprozesses oder erst zur Laufzeit. Die Entscheidung zur Laufzeit bietet den Vorteil, dass eine situationsbewusste Optimierung der Prozedurausführung erfolgen kann, beispielsweise

nach den Kriterien Auslastung, Energieeffizienz oder Verfügbarkeit. Ein Merkmalsystem unterstützt die Zuordnung von konkreten Ausführungseinheiten zu abstrakten Rollen. Die Ausführungseinheiten werden durch Merkmale beschrieben, die den Anforderungen der Rolle genügen müssen.

Die Prozedurbeschreibung kann als Grundlage der Prozedurausführung genutzt werden. Die Steuerung der Ausführung sowohl im Normalfall als auch im Fehlerfall übernimmt der Zustandsautomat der Prozedur. Die Zustände des Zustandsautomats sind fest definiert. Bezüglich der Übergänge zwischen den Zuständen kann es jedoch im konkreten Anwendungsfall Abweichungen geben. Ein Beispiel hierfür ist die Entscheidung, ob eine gehaltene Prozedur abbrechen muss oder neu starten darf. Diese Entscheidung kann nicht generisch getroffen werden. Der Standardfall sieht an dieser Stelle vor, dass alle (demnach auch die Abbruch- und Restart-Transitionen) Transitionen gesperrt sind.

Der Zustandsautomat sorgt für ein deterministisches Ausführungsverhalten der Prozedur. Insbesondere ist die Reihenfolge der Aktionsaufrufe in einem Schritt festgelegt, nicht jedoch die Abfolge der durch die Aktionsaufrufe initiierten Aktivitäten der Ausführungseinheiten. Nicht nur die Reihenfolge der Aktivitätsausführung, sondern auch der Start einer Aktivität in einer Ausführungseinheit ist nicht gesichert. Ausführungseinheiten können Dienstanfragen eigenständig akzeptieren oder verwerfen, beispielsweise wenn ihr Belegungsautomat (vgl. [184]) die Ausführung nicht zulässt. Insbesondere in dezentralen Systemen ist daher eine explizite Abfrage sinnvoll, ob eine Aktivität auch tatsächlich schon gestartet ist. Der Verzicht auf zwingende Antworten beim Dienstaufruf vermeidet aufwendige Konsistenzsicherungs- und Synchronisationsfunktionen. Zudem wird ein Blockieren der Prozedurausführung vermieden, wenn asynchrone Dienstaufrufe ohne Antworten verwendet werden.

Die Ausführung einer Prozedur kann ereignisgetrieben oder getaktet erfolgen. Die Entscheidung für eine der beiden Varianten hängt maßgeblich vom Steuerungssystem ab, das die Prozedur ausführt. Eine SPS auf Basis der IEC 61131-3 führt ihr Programm zyklisch aus, d. h., die Transitionsbedingungen werden getaktet ausgewertet. Menschen hingegen können sowohl getaktet als auch ereignisgetrieben arbeiten. Es ist einerseits vorstellbar, dass sie nach einer festen Zeitvorgabe überprüfen, ob sie die nächsten Schritte einleiten. Andererseits kann ein Mensch auch erst durch das Eintreten eines Ereignisses wieder an die Prozedur erinnert werden, die er ausführen soll. Die Ausführung der Prozedur erfolgt nach der Lock-Step-Semantik. Es kann maximal ein Schrittwechsel in einem Zyklus erfolgen.

Die Kooperation des Referenzmodells mit bestehenden Ausführungseinheiten lässt sich durch die Einführung von zusätzlichen Blöcken ohne großen Aufwand durchführen. Sollten diese Modifikationen unerwünscht sein, kann mithilfe existierender Standards wie NE 150 und PLCopen eine automatische Transformation des generischen Prozedurmodells in steuerungsspezifischen Programmcode erfolgen. Alle Vorteile kann das Referenzmodell erst ausspielen, wenn es in einem Dienstsystem ausgeführt wird. Dieser Punkt ist insbesondere interessant, wenn Industrie 4.0-kompatible Umgebungen im industriellen Umfeld etabliert sind. Im RAMI ist eine Prozedurbeschreibung ein Asset, das seinen eigenen Lebenszyklus hat. Die Prozedurbeschreibung kann auf alle Hierarchieebenen des RAMI oberhalb des Produkts zugreifen. Sie ist entweder in der Geschäftsschicht oder in der Funktionsschicht definiert.

Vor der praktischen Nutzung des Referenzmodells sind einige projektspezifische Festlegungen zu treffen. Zunächst erfolgt die Auswahl des Kommunikationssystems. Es ist eine Abbildung der Dienstaufrufe auf technologische Funktionen des Kommunikationssystems

notwendig. Sobald ein generisches Format zum Aufruf von Diensten in einer Industrie 4.0-Umgebung definiert ist, kann dieser Schritt entfallen. Der zweite Schritt besteht in der Festlegung der Flexibilität bei der Prozedurausführung. Je höher die Flexibilität ist, desto stärker muss das ausführende Steuerungssystem zur Laufzeit rekonfigurierbar sein. Falls das Rollenkonzept verwendet werden soll, muss ein Merkmalsystem in das Laufzeitsystem integriert werden. Das Merkmalsystem überprüft die Erfüllung der spezifizierten Anforderungen der Prozedur an die Ausführungseinheiten.

Der Austausch der Prozeduren zwischen den verschiedenen Planungssystemen und dem Steuerungssystem erfolgt mittels XML-Dateien. Für das vorgestellte Referenzmodell ist eine Darstellung nach dem CAEX-Schema vorgenommen worden. Eine automatische Transformation der CAEX-Darstellung in das OPC UA-Informationsmodell ist möglich. Das Referenzmodell ist jedoch prinzipiell unabhängig von einem konkreten Austauschformat. Durch die Erzeugung von XML-Dateien, in denen nur ein Bezug auf Rollen und nicht auf konkrete Ausführungseinheiten genommen wird, ist die Erstellung einer Datenbank mit wiederverwendbaren Prozeduren möglich. Auf diese Weise leistet das Referenzmodell neben der Durchgängigkeit über den Lebenszyklus der Prozedur einen weiteren Beitrag zur Reduzierung des Aufwands bei der Erstellung von Prozeduren.

A. Zusammenfassung der Analyse

In diesem Kapitel sind die Resultate der Analyse aus Kapitel 3, S. 24, in Tabellenform dargestellt. Jede Tabelle fasst dabei die Aspekte aller betrachteten Sprachen zu einem der Elemente aus Kapitel 3.1, S. 24, zusammen.

A.1. Aufbaumodell

Tabelle A.1.: Zusammenfassung der Aufbaumodelle

	Graph-Elemente		
Sprache	Knoten	Kanten	Sonstiges
AD	Aktionen, Aktivitätsknoten und Objektwerte	Flüsse	Startknoten verpflichtend, Endknoten möglich
BPEL	Aktivitäten, globale Variablen und Schnittstellen	—	—
BPMN	Flussobjekte, Daten, Swimlanes und Artefakte	Verbindungsobjekte	Start- und Endereignisse
EA	Eine endliche Menge von Zuständen	Zustandsübergänge	Mindestens ein Startknoten verpflichtend, Endknoten möglich
EPK	Informationsobjekte, Funktionen und Ereignisse	Kontrollflüsse	—
Grafcet	Schritte und Transitionen	Wirkungslinien	Trennung in Struktur- und Wirkungsteil, mindestens ein Startschritt, Endschritte sind optional
Grafchart	Schritte und Transitionen	Wirklinien	Mindestens ein Startschritt, Endschritte sind optional
K3	Elemente aus den AD und Satelliten-Elemente	Kontrollflüsse	Anfangspunkt und mindestens ein Endpunkt erforderlich

. . . Weiter auf der nächsten Seite . . .

Tabelle A.1.: Zusammenfassung der Aufbaumodelle

	Graph-Elemente		
Sprache	Knoten	Kanten	Sonstiges
PFC	Rezept-Prozedurelemente und explizite Transitionen	Implizite Transitionen und Wirklinien	Element-Synchronisationen, mindestens ein Start- und mindestens ein Endknoten
PLC SC	Zyklusinterne und mehrzyklische Zustände	Deterministische Transitionen mit Priorität	Zusammenfassung von Initialzustand, Auswahlzustand, Gabelungszustand und Kreuzungsknoten als Pseudozustand
PN	Stellen und Transitionen, Stellen besitzen Kapazitäten	Kanten haben Gewichte	Anfangsmarkierung legt initialen Zustand fest
SC	Zustände	Zustandsübergänge	Anfangs- und Endknoten zwingend erforderlich
SFC	Schritte und Transitionen	Steuerungsfluss	Initialschritt zwingend, aber kein Endschritt definiert
SSC	Schritte und Transitionen	Verbindungen	Initialschritt ratsam, mehrere Endschritte möglich

A.2. Hierarchie- und Vernetzungsmodell

Tabelle A.2.: Zusammenfassung der Hierarchie- und Vernetzungsmodelle

Sprache	Vernetzungsregeln	Hierarchieelemente	Nebenläufigkeit
AD	Aktionen werden durch Kontrollflüsse miteinander verbunden	Aktivitätsknoten gruppieren Aktionen, Schwimmbahnen dienen der Übersichtlichkeit	Nebenläufige Pfade möglich, können echt nebenläufig oder sequentiell verzahnt ausgeführt werden
BPEL	—	Strukturierte Aktivitäten fassen Basisaktivitäten zusammen	Durch Flussaktivitäten

. . . Weiter auf der nächsten Seite . . .

Tabelle A.2.: Zusammenfassung der Hierarchie- und Vernetzungsmodelle

Sprache	Vernetzungsregeln	Hierarchieelemente	Nebenläufigkeit
BPMN	Sequenzfluss legt zeitliche Reihenfolge der Aktivitäten fest	Teilprozesse und Schleifen, Pools und Lanes zur organisatorischen Unterteilung	Parallele Gateways und datenbasiert inklusive Gateways
EA	Zustände werden durch Übergänge verbunden	Keine Hierarchie möglich	Nur durch mehrere orthogonale EA modellierbar
EPK	Ereignisse und Funktionen alternierend verbunden, Ereignis ist immer Start und Ende	Über strukturbildende Objekte	Konjunktive und adjunktive Verknüpfungen
Grafcet	Schritte und Transitionen werden im Strukturteil alternierend verbunden	Einschließungen und Makroschritte, von Verwendung wird Anfängern abgeraten	Durch Parallelverzweigung oder mehrere Startschritte
Grafchart	Schritte und Transitionen werden alternierend verbunden	Makroschritte und Prozeduren, Prozeduren unterstützen Parameter und Ausführung in einem separaten Thread	Parallele Verzweigung unterstützt zwei nebenläufige Zweige, kann hintereinander geschaltet werden
K3	Aktivitäten werden durch Kontrollflüsse verbunden	Aggregierte Aktivitäten fassen Aktivitäten mit spezifizierter Reihenfolge zusammen, Blobs sind Aktivitäten ohne spezifizierte Reihenfolge	Verzweigungen und synchrone Zusammenarbeit
PFC	Rezept-Prozedurelemente werden entweder durch implizite Transitionen verbunden oder mittels Wirklinien mit expliziten Transitionen verknüpft, keine Zyklen erlaubt	Durch Prozedurebenen der IEC 61512 auf vier Ebenen beschränkt	Durch Parallelverzweigung

. . . Weiter auf der nächsten Seite . . .

Tabelle A.2.: Zusammenfassung der Hierarchie- und Vernetzungsmodelle

Sprache	Vernetzungsregeln	Hierarchieelemente	Nebenläufigkeit
PLC SC	Die Zustände werden durch die Zustandsübergänge verbunden, zusätzlich können Unterbrechungstransitionen verwendet werden	Gegenüber SC dürfen Super-Zustände keine eigenen Aktionen haben und müssen einen definierten Startpunkt haben	Gegenüber SC muss jede orthogonale Region einen Endzustand haben
PN	Stellen und Transitionen müssen alternierend verknüpft werden	Keine Hierarchie möglich	Gleichzeitig mehrere aktive Schritte und unabhängige Schaltbarkeit der Transitionen
SC	Die Zustände werden durch die Zustandsübergänge verbunden	Super-Zustände mit Unter-Zuständen, Zustandsübergänge sowohl von Unter-Zuständen als auch von Super-Zuständen aus möglich	Orthogonale Zustände sind spezielle Super-Zustände, Synchronisation durch gemeinsame Übergangsbedingungen
SFC	Schritte und Transitionen werden über den Steuerungsfluss in der POE verknüpft	Aufruf anderer SFC, wird nicht empfohlen, Integration in Funktionsbausteine	Durch Simultanketten
SSC	Schritte und Transitionen werden über Verbindungen im Ausführungsrahmen verknüpft	Unterprozeduren mit eigenem Ausführungsrahmen und innerhalb eines Ausführungsrahmens möglich	Nur durch Aufruf mehrerer Unterprozeduren

A.3. Abstraktions- und Zuordnungsmodell

Tabelle A.3.: Zusammenfassung der Abstraktions- und Zuordnungsmodelle

Sprache	Typkonzept	Sonstiges
AD	Bibliothek mit Aktivitätsknoten vorgesehen	Top-Down-Entwurfsprozess empfohlen
BPEL	Abstrakte und ausführbare Prozeduren, Dienste können sowohl auf Typebene als auch auf Instanzebene orchestriert werden	—

. . . Weiter auf der nächsten Seite . . .

Tabelle A.3.: Zusammenfassung der Abstraktions- und Zuordnungsmodelle

Sprache	Typkonzept	Sonstiges
BPMN	Abstrakte und ausführbare Prozeduren, globale Teilprozeduren können mehrfach aufgerufen werden	—
EA	Kein Typkonzept vorhanden	—
EPK	Abstraktionsebene mit Typen und Ausprägungsebene	—
Grafcet	Wiederverwendung von Grafcets als Einschließung	—
Grafchart	Prozeduren können durch Parameter angepasst werden	—
K3	Kein Typkonzept vorhanden	—
PFC	Bibliothek für wesentliche Ablaufereignisse	Abstraktionsschichten über Rezeptmodell der IEC 61512
PLC SC	Kein Typkonzept vorhanden	Automatische SPS-Codegenerierung möglich
PN	Kein Typkonzept vorhanden	Interpretationen legen die semantische Bedeutung von Stellen und Transitionen fest
SC	Kein Typkonzept vorhanden	SC sind eine Sicht auf das UML-Gesamtmodell
SFC		Top-Down-Entwurfsprozess und Bottom-Up-Implementierung empfohlen
SSC	Typkonzept analog zu Funktionsbausteinen	Zunächst Vorgabe der Ein- und Ausgänge, anschließend Prozedurentwurf ratsam

A.4. Aktions- und Aktivitätenmodell

Tabelle A.4.: Zusammenfassung der Aktions- und Aktivitätenmodelle

Sprache	Aktion	Aktionsausführung	Übergangsbedingung
AD	Keine einheitliche Syntax festgelget	Bei der Aktivierung eines Schritts, Aktionen können Vor- und Nachbedingung haben	Keine einheitliche Syntax festgelget

. . . Weiter auf der nächsten Seite . . .

Tabelle A.4.: Zusammenfassung der Aktions- und Aktivitätenmodelle

Sprache	Aktion	Aktionsausführung	Übergangsbedingung
BPEL	Nachrichtenbasierte Dienstaufrufe, Manipulation der Variablen	Dienste durch Aufrufobjekte gestartet	Bei Alternativverzweigungen in Xpath angegeben
BPMN	Aktivitäten haben keine feste Syntax, werden durch Menschen oder Automaten ausgeführt	Bei der Aktivierung eines Schritts	Zwischenereignisse, wenn gewartet werden soll
EA	Nur in Erweiterungen nach Moore und Mealy (Transduktoren) möglich	Je nach Typ bei Zustand oder bei Zustandsübergang	Durch Eingabesymbole formuliert
EPK	Interaktion mit der Umgebung über Informationsobjekte	Bei der Aktivierung eines Schritts	Keine Syntax für Bedingungen festgelegt
Grafcet	Kontinuierlich wirkende und gespeichert wirkende Aktionen im Wirkungsteil	Bei der Aktivierung eines Schritts	Boolesche Ausdrücke, die aus Variablen und internen Ereignissen zusammengesetzt sind
Grafchart	Aktionen sind Aufrufe in G2 oder Java, es werden Dienstaufrufe unterstützt	Durch Präfix gesteuert, in Unterarbeitsbereich eines Schritts angeordnet	Wächterbedingung aus booleschen Ausdrücken und Ereignissen
K3	Interaktion mit der Umgebung über Informationsobjekte	Bei der Aktivierung eines Schritts	Keine Syntax für Bedingungen festgelegt
PFC	Einrichtungssteuerung wird aufgerufen, ist selber nicht Bestandteil der Sprache	Aktionsaufrufe in Schritten und Transitionen möglich	Keine Sprache für Bedingungen festgelegt, implizite Transitionen haben keine Bedingung
PLC SC	Aktionen entsprechen denen im SFC, Aktivitäten haben Initialisierungs-Eingang und Beendet-Ausgang	Emulation von Ereignissen in einer SPS	Wie bei SC

... Weiter auf der nächsten Seite ...

Tabelle A.4.: Zusammenfassung der Aktions- und Aktivitätenmodelle

Sprache	Aktion	Aktionsausführung	Übergangsbedingung
PN	In der Interpretation SIPN durch die Funktion q_P möglich	In SIPN im Schritt	In der Interpretation SIPN durch die Funktion q_T möglich
SC	Unterschied zwischen Aktionen und Aktivitäten, Aktionen sind zeitlos, Aktivitäten haben Dauer, Aktionen starten bzw. stoppen Aktivitäten	Aktionen können in einem Zustand und während eines Übergangs ausgeführt werden	Zustandsübergänge haben einen Ereignistrigger und eine überwachte Bedingung
SFC	Aktionen in Programmiersprache der IEC 61131-3	Durch Aktionsbestimmungszeichen gesteuert, in Aktionsblock eines Schritts angeordnet	Bedingung in Programmiersprache der IEC 61131-3
SSC	Setzen von Ausgangsvariablen, Setzen eines Eingangs lokaler Funktionsbausteine oder Start lokaler Funktionsbausteine möglich	Beim Betreten/Verlassen eines Schritts oder zyklisch, während der Schritt aktiv ist	Durch Erstellen eines Funktionsbausteinnetzwerks

A.5. Ausführungssteuerungsmodell

Tabelle A.5.: Zusammenfassung der Ausführungssteuerungsmodelle

Sprache	Ausführungsmodell	Verhalten im Fehlerfall und bei Konflikten
AD	Token-basiertes Ausführungsmodell	Definition von Abbrüchen für Unterbrechungsbereiche möglich
BPEL	Synchrone und asynchrone Ausführung. Aktivitäten werden der Reihe nach ausgeführt	Unterscheidung zwischen fachlichen und technischen Fehlern, technische Fehler können durch explizite Abbrüche und Fehlerbehandlungsmechanismen behandelt werden

... Weiter auf der nächsten Seite ...

Tabelle A.5.: Zusammenfassung der Ausführungssteuerungsmodelle

Sprache	Ausführungsmodell	Verhalten im Fehlerfall und bei Konflikten
BPMN	Token-basiertes Ausführungsmodell	Ersteller muss Verklemmung vermeiden, Standard-Pfade sind geeignetes Hilfsmittel
EA	Weiterschaltung durch aktuellen Zustand und Eingabesymbol festgelegt	Undefiniertes Verhalten im Fehlerfall
EPK	Keine Beschreibung der Ausführung	Kombination mit ARIS zur Ausführung notwendig
Grafcet	Transitionen feuern, wenn alle Schritte vor der Transition aktiv sind und die Bedingung *wahr* ist, es können mehrere Transitionen gleichzeitig feuern	Verhinderung von Konflikten ist Aufgabe des Entwicklers
Grafchart	Zyklisches Ausführungsmodell, kein Durchschalten möglich	Bei Konflikten in einer Alternativverzweigung werden mehrere Pfade aktiviert, Makroschritte und Prozeduren haben Abbruchtransition, interner Zustand wird dann gespeichert und beim nächsten Aufruf dort fortgesetzt
K3	Analog zu AD, Ausführungsreihenfolge von Aktivitäten in Blobs wird zur Laufzeit festgelegt	Optionale und verbotene Aktivitäten
PFC	Ausführung durch Zustandsautomaten gesteuert, selbstbeendende Schritte bei impliziten Transitionen, Synchronisationen beeinflussen Abläufe	Ausnahmebehandlung durch Zustandsautomaten
PLC SC	Formale Ausführungslogik in UPPAAL	Fehlerbehandlung durch ausführende SPS
PN	Weiterschaltung, sobald im Vorbereich einer Transition genügend Marken liegen und im Nachbereich der Transtion Platz für die Aufnahme der Marken ist	Eine Verklemmung ist möglich und kann nicht automatisch behoben werden

. . . Weiter auf der nächsten Seite . . .

Tabelle A.5.: Zusammenfassung der Ausführungssteuerungsmodelle

Sprache	Ausführungsmodell	Verhalten im Fehlerfall und bei Konflikten
SC	Zustandsübergänge feuern, wenn sowohl die Bedingung *wahr* ist als auch das Ereignis getriggert wurde, Aktionen können bei Betreten und Verlassen eines Schritts ausgeführt werden, es kann nur ein Zustand aktiv sein	Tritt ein Ereignis ein, das im aktuellen Zustand nicht modelliert ist, so verharrt das System im Zustand
SFC	Transitionen feuern, wenn alle Schritte vor der Transition aktiv sind und die Bedingung *wahr* ist, Aktionsausführung durch internen Funktionsbaustein gesteuert, Maximal-Progress-Vorgehen oder Lock-Step-Vorgehen möglich, erst Transitionsauswertung, dann Aktionsausführung oder umgekehrt	Unsichere und verklemmende Abläufe möglich, bei Fehlern wird Ablauf gestoppt und es sind mehrere Korrekturverfahren möglich
SSC	Formale Ausführungslogik in UPPAAL	Keine Fehlerbehandlung im Modell vorgesehen

B. Prozeduren der Akteure bei einer Ventilwartung

Im Folgenden werden die Prozeduren der Akteure vorgestellt, die im Beispiel der Ventilreparatur (vgl. Kapitel 5.3, S. 97) miteinander agieren.

Zunächst wird die Prozedur des Logistikers in Abbildung B.1 gezeigt. Sobald dieser die Aufforderung zum Ausliefern erhalten hat (***RR1***), gibt er sich selber die Aufforderung, das gewünschte Ersatzteil an den gewünschten Ort zu liefern (***A1***). Sobald das Ersatzteil am Bestimmungsort ist (***RR2***), informiert er den Wartungstechniker (***A2***).

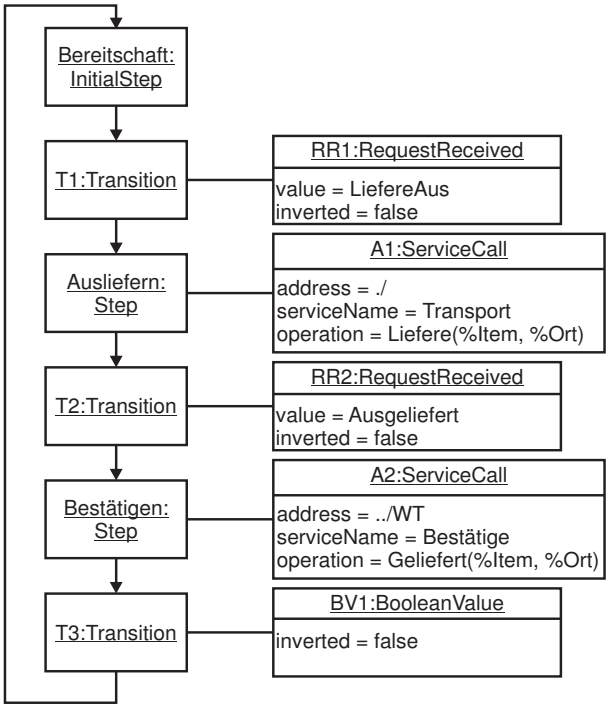

Abbildung B.1.: Detaillierte Prozedur des Logistikers

Der Produktionsplaner (vgl. Abbildung B.2) weicht von seiner Standard-Planung ab,

wenn er die Aufforderung zum Vorbereiten einer Wartungsmaßnahme erhält (***RR1***). Anhand des prognostizierten Liefertermins plant er die Maßnahme in die Produktion ein (***A1***) und informiert den Logistiker über die Warenlieferung (***A2***), sobald das Ersatzteil eingetroffen ist (***RR2***). Nach erfolgreicher Durchführung der Wartungsmaßnahme (***RR3***) geht er wiederum zur Standardplanung über.

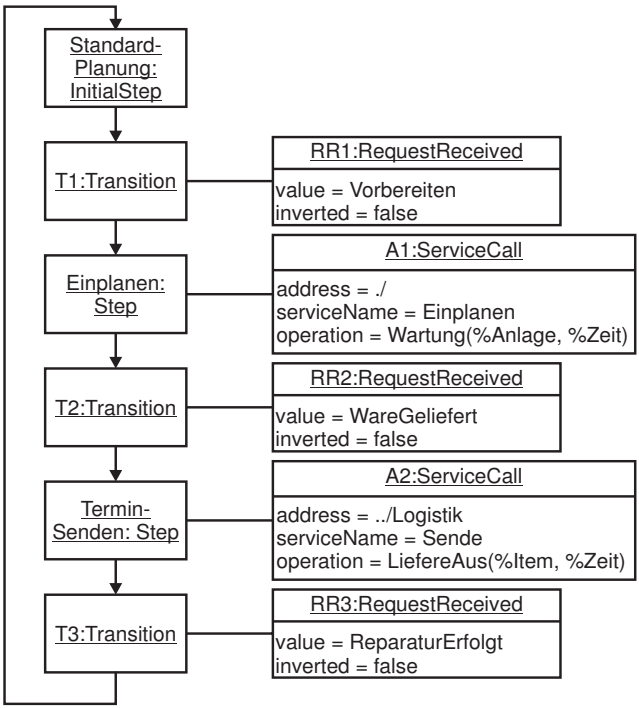

Abbildung B.2.: Detaillierte Prozedur des Produktionsplaners

Der Einkäufer (vgl. Abbildung B.3) startet seine Tätigkeit, sobald er die Aufforderung zu einer Bestellung erhält (***RR1***). Anschließend ermittelt er einen Lieferanten (***A1***) und bestellt dort das Ersatzteil (***A2***). Der Lieferant teilt dem Einkäufer einen Liefertermin mit (***RR3***), den dieser an den Produktionsplaner weiterleitet (***A3***). Sobald das Ersatzteil geliefert ist (***RR4***), bestätigt der Einkäufer dies dem Produktionsplaner (***A4***).

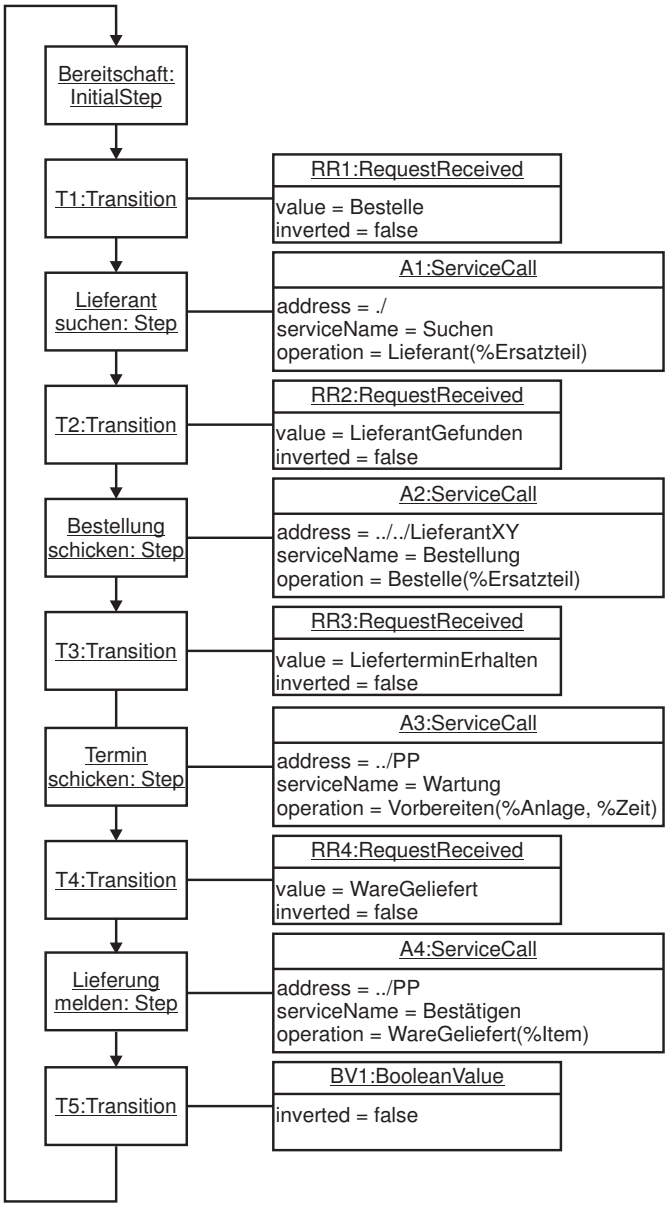

Abbildung B.3.: Detaillierte Prozedur des Einkäufers

Der Anlagenfahrer hat in seiner Prozedur, die in Abbildung B.4 abgebildet ist, drei Alternativen, wenn er den Prozess beobachtet (***BeobachteProzess***). Er kann

- eine Fehlersuche beauftragen, wenn der Durchfluss in der Teilanlage TA2 zu niedrig ist (***SC1***),
- das Betreten der Anlage freigeben, wenn er dazu aufgefordert wird (***RR3***) und
- den Prozess in einen Zustand fahren, der die Reparatur ermöglicht (***RR1***).

Die einzelnen Abläufe zur Durchführung der drei Alternativen sind der Übersichtlichkeit halber in Makroschritten gekapselt.

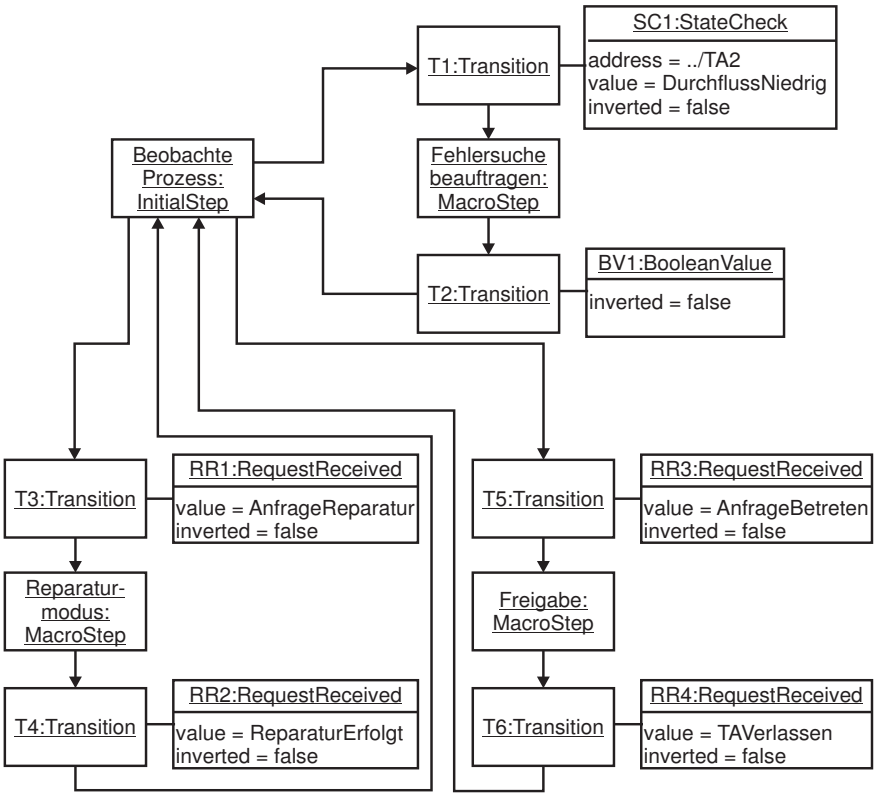

Abbildung B.4.: Prozedur des Anlagenfahrers mit Makroschritten

Auch der Wartungstechniker hat in seiner Prozedur, die in Abbildung B.5 abgebildet ist, drei Alternativen. Er kann

- eine Fehlersuche durchführen (***Fehlersuche***), wenn er dazu beauftragt wird (***RR1***),
- eine Bestellung (***Bestellung***) ausführen (***RR2***) und
- eine Reparatur (***Reparatur***) durchführen (***RR3***).

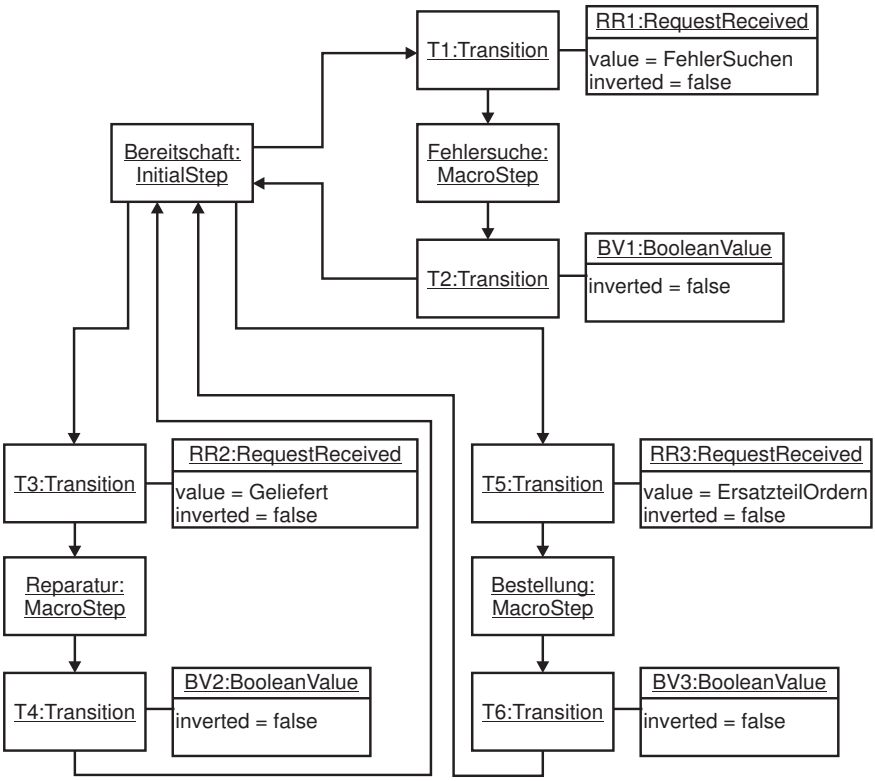

Abbildung B.5.: Prozedur des Wartungstechnikers mit Makroschritten

C. Kompabilität zu bestehenden Beschreibungssprachen

C.1. Umschreibung der Aktionsbestimmungszeichen

Die folgende Tabelle beschreibt die Umsetzung der Aktionsbestimmungszeichen eines SFCs mithilfe des Referenzmodells. Zu beachten ist, dass Aktionen im Referenzmodell die Ausführungseinheiten triggern. Eine Aktion nach der IEC 61131-3 kann jedoch eine zeitliche Dauer haben und ist nach der Begriffswelt des Referenzmodells eine Aktivität. Daher wird in der rechten Spalte der Tabelle von Start- und Stop-Aktionen gesprochen, die die jeweilige Aktivität steuern.

Tabelle C.1.: Umschreibung der Aktionsbestimmungszeichen

Zeichen	Beschreibung	Modellierung mit Referenzmodell
Kein	Nicht gespeichert. Die zugehörige Aktion wird ausgeführt, während der Schritt aktiv ist.	Zuordnung der Start-Aktion zu einem Schritt und Zuordnung der Stop-Aktion zu allen Folgeschritten.
N	Nicht gespeichert. Die zugehörige Aktion wird ausgeführt, während der Schritt aktiv ist.	Zuordnung der Start-Aktion zu einem Schritt und Zuordnung der Stop-Aktion zu allen Folgeschritten.
R	Vorrangiges Rücksetzen. Die zugehörige Aktion wird nicht mehr ausgeführt.	Zuordnung der Stop-Aktion zu einem Schritt.
S	Setzen (gespeichert). Die zugehörige Aktion wird ausgeführt, bis sie rückgesetzt wird.	Zuordnung der Start-Aktion zu einem Schritt, Standardfall im Referenzmodell.
L	Zeitbegrenzt. Die zugehörige Aktion wird ausgeführt, bis entweder die Zeitspanne abgelaufen ist oder der Schritt deaktiviert wird.	Zuordnung der Start-Aktion zu einem Schritt und Zuordnung einer expliziten Stop-Aktion zu allen Folgeschritten. Einfügen eines zusätzlichen Schrittes, der über eine Transition mit Zeitüberwachung erreicht wird. Zuordnung der Stop-Aktion zu diesem Schritt.

... Weiter auf der nächsten Seite ...

Tabelle C.1.: Umschreibung der Aktionsbestimmungszeichen

Zeichen	Beschreibung	Modellierung mit Referenzmodell
D	Zeitverzögert. Die zugehörige Aktion wird nach Ablauf der Zeitspanne ausgeführt, bis der Schritt deaktiviert wird.	Einfügen eines zusätzlichen Schrittes, der über eine Transition mit Zeitüberwachung erreicht wird. Zuordnung der Start-Aktion zu diesem Schritt. Zuordnung der Stop-Aktion zu allen Folgeschritten.
P	Impuls (Flanke). Die zugehörige Aktion wird bei der Aktivierung und bei der Deaktivierung des Schritts einmal ausgeführt.	Zusätzliche Zuordnung der Start-Aktion zu allen Folgeschritten.
SD	Gespeichert und zeitverzögert. Die zugehörige Aktion wird nach Ablauf der Zeitspanne ausgeführt, bis sie rückgesetzt wird.	Einfügen eines zusätzlichen Schrittes, der über eine Transition mit Zeitüberwachung erreicht wird. Zuordnung der Start-Aktion zu diesem Schritt.
DS	Verzögert und gespeichert. Die zugehörige Aktion wird nach Ablauf der Zeitspanne ausgeführt, bis sie rückgesetzt wird, es sei denn, der Schritt wird vor Ablauf der Zeitspanne deaktiviert.	Einfügen eines zusätzlichen Schrittes, der über eine Transition mit Zeitüberwachung erreicht wird. Zuordnung der Start-Aktion zu diesem Schritt.
SL	Gespeichert und zeitbegrenzt. Die zugehörige Aktion wird ausgeführt, bis die Zeitspanne abgelaufen ist.	Zuordnung der Start-Aktion zu einem Schritt. Einfügen eines zusätzlichen Schrittes, der über eine Transition mit Zeitüberwachung erreicht wird. Zuordnung der entsprechenden Stop-Aktion zu diesem Schritt. Direkte Rückkehr zum ursprünglichen Schritt
P1	Puls (steigende Flanke). Die zugehörige Aktion wird bei der Aktivierung des Schritts einmal ausgeführt.	Zuordnung der Start-Aktion zu einem Schritt.
P0	Puls (fallende Flanke). Die zugehörige Aktion wird bei der Deaktivierung des Schritts einmal ausgeführt.	Zuordnung der Start-Aktion zu allen Folgeschritten.

C.2. Priorität der Alternativverzweigungen

Im Referenzmodell wird ausschließlich die Verwendung von sich gegenseitig ausschließenden Transitionsbedingungen unterstützt (vgl. Kapitel 4.2.2, S. 70). In manchen Beschreibungssprachen sind jedoch auch andere Möglichkeiten zur Priorisierung möglich, die Auswertung von links nach rechts oder die explizite Vergabe von Prioritäten. Im Folgenden wird der Fall der Auswertung von links nach rechts betrachtet, der in Abbildung C.1 dargestellt ist. Der Fall der expliziten Vergabe von Prioritäten muss nicht gesondert betrachtet werden, da hierbei immer eine grafische Anordnung von links nach rechts möglich ist und wiederum der erste Fall betrachtet werden kann.

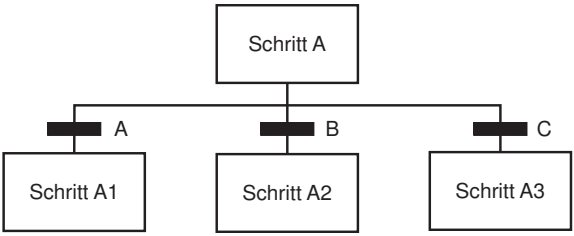

Abbildung C.1.: Priorisierung der Alternativverzweigung von links nach rechts

Wird der Ausschnitt aus einer Prozedur in Abbildung C.1 mit dem Referenzmodell modelliert, ergeben sich die Instanzen in Abbildung C.2.

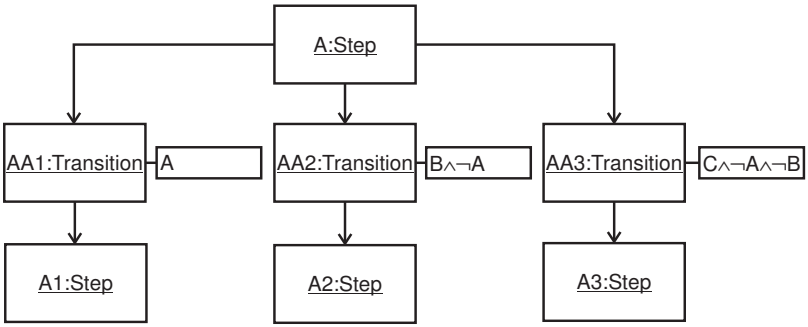

Abbildung C.2.: Modellierung der Alternativverweigung im Referenzmodell

Die entsprechenden Transitionsbedingungen `A-C` sind in Abbildung C.2 der Übersichtlichkeit wegen als logischer Ausdruck und nicht als zusammengesetzte **Condition** dargestellt. Es ist offensichtlich, dass nur die Transition *AA1* schalten kann, wenn `A` *wahr* ist und *AA3* nur schalten kann, wenn `A` und `B` *falsch* sind. *AA2* kann nur schalten, wenn `A` *falsch* ist und schaltet ebenfalls, wenn `C` *wahr* ist.

C.3. Modellierung von Do und Exit

Aktionen können im Referenzmodell ausschließlich bei der Aktivierung eines Schrittes ausgeführt werden. Verschiedene Sprachen, beispielsweise SC und SSC, unterstützen jedoch auch die Aktionsausführung beim Verlassen eines Schritts (Abbildung C.3 links) und die zyklische Ausführung von Aktionen, solange ein Schritt aktiv ist (Abbildung C.4 links). Die Aktionsausführung beim Verlassen eines Schritts lässt sich durch die Zuordnung der Aktion zu den Folgeschritten des zu verlassenden Schritts modellieren (Abbildung C.3 mittig). Ist die Reihenfolge zwischen der Ausführung der Aktionen beim Verlassen des alten und beim Betreten des neuen Schritts wichtig, muss ein zusätzlicher Schritt eingefügt werden (Abbildung C.3 rechts).

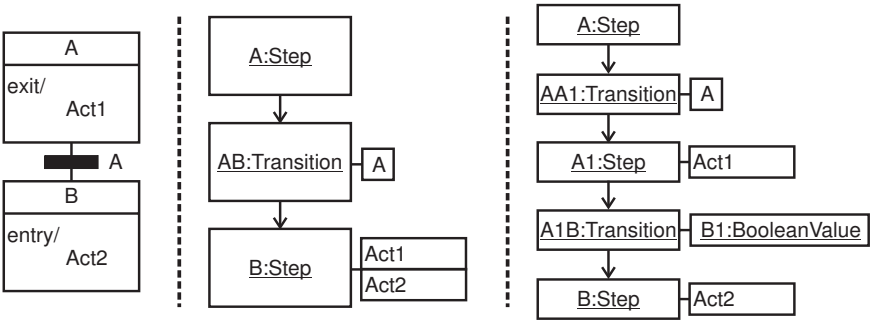

Abbildung C.3.: Modellierung von Aktionsaufrufen beim Verlassen eines Schritts

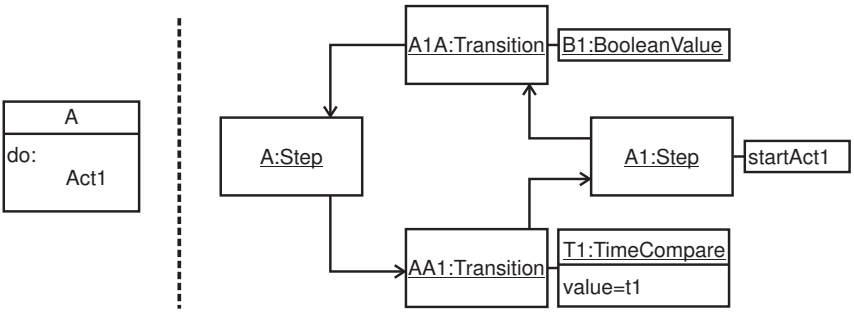

Abbildung C.4.: Modellierung wiederkehrender Aktivitäten

Zur Modellierung von Do-Aktivitäten muss die Start-Aktion der Aktivität zu einem zusätzlichen Schritt hinzugefügt werden (Abbildung C.4 links). Dieser zusätzliche Schritt wird durch die Zeitvergleich-Transition und die Schleife zyklisch aufgerufen, solange nicht

eine andere Transition des Schritts **A** feuert. Durch Anpassung der Zeit *t1* von **T1** kann die Frequenz der Aufrufe eingestellt werden.

C.4. Verarbeitung von Messwerten und Setzen von Stellgrößen

Die Abfrage von Messwerten ist im Referenzmodell nicht vorgesehen. Daher muss in den Ausführungseinheiten eine Diskretisierung der Messwerte zu Zuständen vorgenommen werden. Die Diskretisierung erfolgt so, dass alle von der steuernden Prozedur verwendeten Zustandsabfragen durchgeführt werden können. In Abbildung C.5 ist eine Diskretisierung beispielhaft dargestellt. In diesem Beispiel wird der Behälter in drei Bereiche eingeteilt, viel, normal und wenig Inhalt. Hierzu sind zwei Größen x_1 und x_2 festgelegt, die die Bereiche abgrenzen. Die Größen x_1 und x_2 müssen nicht im Zusammenhang mit den Grenzwerten des Behälters stehen.

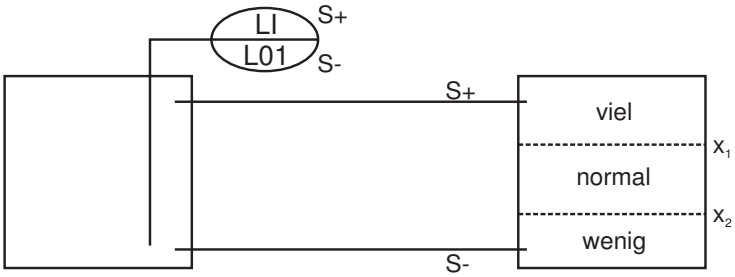

Abbildung C.5.: Diskretisierung eines Behälterfüllstands

Die Realisierung der Diskretisierung als Funktionsbausteinnetzwerk erfolgt mithilfe von Vergleichsoperatoren, logischen Operatoren und Schaltern, wie in Abbildung C.6 dargestellt. Je nach aktuellem Messwert wird eines der Eingangssignale *act1* bis *act3* des Bausteins **3Switch** *wahr* und der entsprechende Wert *val1* bis *val3* auf den Ausgang *out* gelegt, der an den Zustandsausgang *WS* (vgl. Kapitel 5.1.1, S. 90) weitergegeben wird.

Abbildung C.7 zeigt am Beispiel eines Stellventils die dienstbasierte Ansteuerung einer Funktion. Das Ventil bietet den Dienst Ansteuerung mit den Operationen OPEN und CLOSE an. Die Operation OPEN benötigt die Öffnung des Ventils in Prozent als Parameter. Die Operation mit optionalem Parameter wird als Nachricht an die Einzelsteuerung übergeben. In einem ersten Schritt zerlegt der Baustein **Split** die Nachricht in die Operation *cmd* und, falls vorhanden, den Parameter *val*. Der Baustein **CMP** überprüft, ob die Operation OPEN aufgerufen wurde. In diesem Fall wird eine Eins ausgegeben und der Baustein **Switch** schaltet auf den Eingang *val2*, der den Öffnungsgrad beinhaltet. Andernfalls gibt **CMP** eine Null aus.

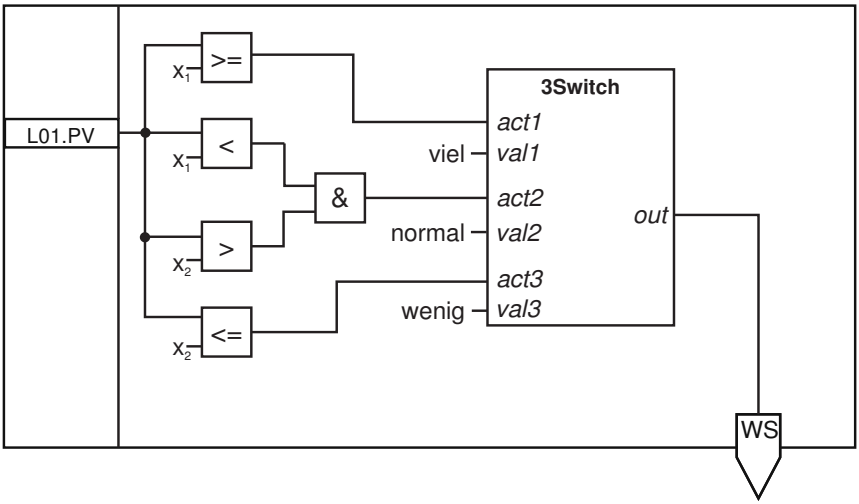

Abbildung C.6.: Funktionsbausteinnetzwerk zur Diskretisierung eines Behälterfüllstands

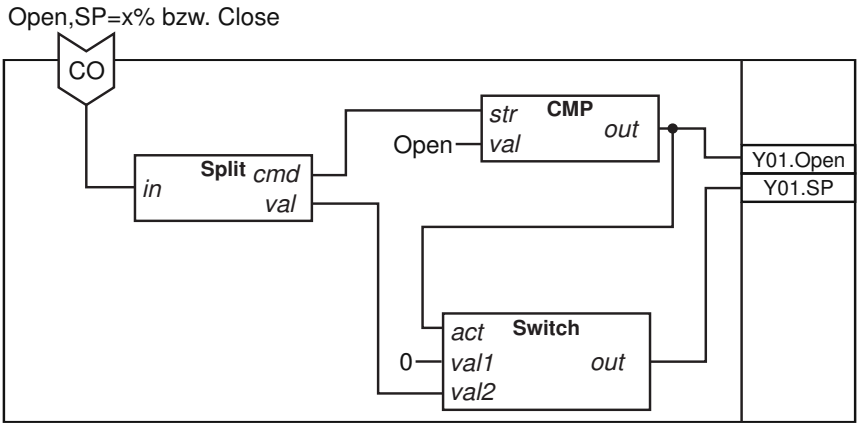

Abbildung C.7.: Funktionsbausteinnetzwerk zur Ansteuerung eines Stellventils

D. XML-Repräsentation des Referenzmodells

```
<?xml version="1.0" encoding="UTF-8"?>
<CAEXFile
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".\CAEX_ClassModel.xsd"
xmlns:iec61360="http://std.iec.ch/cdd/iec61360"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
SchemaVersion="2.15" FileName="ProcedureModel.xml">
  <Version>V1.0.0 (2016)</Version>
  <Copyright>(C) 2016 Andreas Schueller</Copyright>
  <!-- Defining the interfaces of the procedure elements-->
  <InterfaceClassLib Name="ProcedureBaseICL">
    <InterfaceClass Name="ProcedureInterface"/>
    <!-- Interfaces of a step -->
    <InterfaceClass Name="StepIn" RefBaseClassPath="ProcedureBaseICL/
        ProcedureInterface"/>
    <InterfaceClass Name="StepOut" RefBaseClassPath="ProcedureBaseICL/
        ProcedureInterface"/>
    <!-- Interfaces of a transition -->
    <InterfaceClass Name="TransitionIn" RefBaseClassPath="
        ProcedureBaseICL/ProcedureInterface"/>
    <InterfaceClass Name="TransitionOut" RefBaseClassPath="
        ProcedureBaseICL/ProcedureInterface"/>
  </InterfaceClassLib>
  <!-- Defining a role class for the state machine-->
  <RoleClassLib Name="ProcedureBaseRCL">
    <RoleClass Name="StateMachine">
      <Attribute Name="inRequest" AttributeDataType="xs:string">
        <Constraint Name="RequestType">
          <NominalScaledType>
            <RequiredValue>Start</RequiredValue>
            <RequiredValue>Restart</RequiredValue>
            <RequiredValue>Lock</RequiredValue>
            <RequiredValue>Unlock</RequiredValue>
            <RequiredValue>Abort</RequiredValue>
            <RequiredValue>Reset</RequiredValue>
          </NominalScaledType>
        </Constraint>
      </Attribute>
      <Attribute Name="curState" AttributeDataType="xs:string"/>
    </RoleClass>
```

```
</RoleClassLib>
<!-- Defining the procedure elements-->
<SystemUnitClassLib Name="ProcedureBaseSUCL">
  <!-- Basic Elements -->
  <SystemUnitClass Name="ProcedureElement"/>
  <SystemUnitClass Name="ExecutionFrame" RefBaseClassPath="
      ProcedureBaseSUCL/ProcedureElement">
    <Attribute Name="parameter" AttributeDataType="xs:anyType"/>
    <Attribute Name="inRequest" AttributeDataType="xs:string"/>
    <Attribute Name="curStep" AttributeDataType="xs:string"/>
    <InternalElement Name="InitialStep" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/InitialStep"/>
    <InternalElement Name="StateMachine">
      <RoleRequirements RefBaseRoleClassPath="ProcedureBaseRCL/
          StateMachine"/>
    </InternalElement>
  </SystemUnitClass>
  <SystemUnitClass Name="Step" RefBaseClassPath="ProcedureBaseSUCL/
      ProcedureElement">
    <Attribute Name="activationTime" AttributeDataType="xs:time"/>
    <Attribute Name="isActive" AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
    <Attribute Name="name" AttributeDataType="xs:string"/>
    <Attribute Name="timeActive" AttributeDataType="xs:integer" Unit
        ="iec61360:UAA899"/>
  </SystemUnitClass>
  <SystemUnitClass Name="Transition" RefBaseClassPath="
      ProcedureBaseSUCL/ProcedureElement">
    <Attribute Name="name" AttributeDataType="xs:string"/>
    <Attribute Name="enabled" AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
    <Attribute Name="abort" AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
    <Attribute Name="restart" AttributeDataType="xs:integer">
      <DefaultValue>0</DefaultValue>
    </Attribute>
    <ExternalInterface Name="I1" RefBaseClassPath="ProcedureBaseICL/
        TransitionIn"/>
    <ExternalInterface Name="O1" RefBaseClassPath="ProcedureBaseICL/
        TransitionOut"/>
    <InternalElement Name="Condition" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/Condition"/>
  </SystemUnitClass>
  <SystemUnitClass Name="ElementaryStep" RefBaseClassPath="
      ProcedureBaseSUCL/Step"/>
  <SystemUnitClass Name="InitialStep" RefBaseClassPath="
      ProcedureBaseSUCL/ElementaryStep">
```

```
  <ExternalInterface Name="O1" RefBaseClassPath="ProcedureBaseICL/
      StepOut"/>
</SystemUnitClass>
<SystemUnitClass Name="FinalStep" RefBaseClassPath="
    ProcedureBaseSUCL/ElementaryStep">
  <Attribute Name="abort" AttributeDataType="xs:boolean">
    <DefaultValue>false</DefaultValue>
  </Attribute>
  <ExternalInterface Name="I1" RefBaseClassPath="ProcedureBaseICL/
      StepIn"/>
</SystemUnitClass>
<!-- Hierarchy Elements-->
<SystemUnitClass Name="MacroStep" RefBaseClassPath="
    ProcedureBaseSUCL/Step">
  <InternalElement Name="MacroInitial" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/InitialStep"/>
  <InternalElement Name="MacroFinal" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/FinalStep"/>
</SystemUnitClass>
<SystemUnitClass Name="PMacroStep" RefBaseClassPath="
    ProcedureBaseSUCL/MacroStep">
  <InternalElement Name="Fork" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Fork"/>
  <InternalElement Name="Join" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Join"/>
  <InternalLink Name="IL1" RefPartnerSideA="MacroInitial:O1"
      RefPartnerSideB="Fork:I1"/>
  <InternalLink Name="IL2" RefPartnerSideA="Join:O1"
      RefPartnerSideB="MacroFinal:I1"/>
</SystemUnitClass>
<SystemUnitClass Name="Fork" RefBaseClassPath="ProcedureBaseSUCL/
    ProcedureElement">
  <ExternalInterface Name="I1" RefBaseClassPath="ProcedureBaseICL/
      TransitionIn"/>
  <ExternalInterface Name="O1" RefBaseClassPath="ProcedureBaseICL/
      TransitionOut"/>
  <ExternalInterface Name="O2" RefBaseClassPath="ProcedureBaseICL/
      TransitionOut"/>
</SystemUnitClass>
<SystemUnitClass Name="Join" RefBaseClassPath="ProcedureBaseSUCL/
    ProcedureElement">
  <ExternalInterface Name="I1" RefBaseClassPath="ProcedureBaseICL/
      TransitionIn"/>
  <ExternalInterface Name="I2" RefBaseClassPath="ProcedureBaseICL/
      TransitionIn"/>
  <ExternalInterface Name="O1" RefBaseClassPath="ProcedureBaseICL/
      TransitionOut"/>
</SystemUnitClass>
<!--Action and condition elements-->
<SystemUnitClass Name="Action" RefBaseClassPath="ProcedureBaseSUCL
```

```
    /ProcedureElement"/>
  <SystemUnitClass Name="ServiceCall" RefBaseClassPath="
    ProcedureBaseSUCL/Action">
    <Attribute Name="adress" AttributeDataType="xs:string"/>
    <Attribute Name="serviceName" AttributeDataType="xs:string"/>
    <Attribute Name="operation" AttributeDataType="xs:string"/>
    <Attribute Name="parameter" AttributeDataType="xs:anyType"/>
  </SystemUnitClass>
  <SystemUnitClass Name="ProcedureCall" RefBaseClassPath="
    ProcedureBaseSUCL/Action">
    <Attribute Name="adress" AttributeDataType="xs:string"/>
    <Attribute Name="request" AttributeDataType="xs:string"/>
    <Attribute Name="parameter" AttributeDataType="xs:anyType"/>
  </SystemUnitClass>
  <SystemUnitClass Name="Condition" RefBaseClassPath="
    ProcedureBaseSUCL/ProcedureElement">
    <Attribute Name="curValue" AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
    <Attribute Name="listOfOperator" AttributeDataType="xs:anyType"/
      >
  </SystemUnitClass>
  <SystemUnitClass Name="LogicalTerm" RefBaseClassPath="
    ProcedureBaseSUCL/ProcedureElement">
    <Attribute Name="inverted" AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
    <Attribute Name="retValue" AttributeDataType="xs:boolean">
      <DefaultValue>false</DefaultValue>
    </Attribute>
    <Attribute Name="index" AttributeDataType="xs:integer">
      <DefaultValue>1</DefaultValue>
    </Attribute>
  </SystemUnitClass>
  <SystemUnitClass Name="RequestReceived" RefBaseClassPath="
    ProcedureBaseSUCL/LogicalTerm">
    <Attribute Name="value" AttributeDataType="xs:string"/>
  </SystemUnitClass>
  <SystemUnitClass Name="BooleanValue" RefBaseClassPath="
    ProcedureBaseSUCL/LogicalTerm"/>
  <SystemUnitClass Name="StateCheck" RefBaseClassPath="
    ProcedureBaseSUCL/LogicalTerm">
    <Attribute Name="adress" AttributeDataType="xs:string"/>
    <Attribute Name="value" AttributeDataType="xs:string"/>
  </SystemUnitClass>
  <SystemUnitClass Name="TimeCompare" RefBaseClassPath="
    ProcedureBaseSUCL/LogicalTerm">
    <Attribute Name="value" AttributeDataType="xs:integer" Unit="
      iec61360:UAA899"/>
  </SystemUnitClass>
```

```
</SystemUnitClassLib>
<SystemUnitClassLib Name="ProcedureStateMachine">
  <SystemUnitClass Name="DefaultStateMachine">
    <Attribute Name="inRequest"/>
    <Attribute Name="curState"/>
    <!-- States of the state machine -->
    <InternalElement Name="Aborted" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/ElementaryStep">
      <ExternalInterface Name="I1" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="O1" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
    </InternalElement>
    <InternalElement Name="Idle" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/ElementaryStep">
      <ExternalInterface Name="I1" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="I2" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="O1" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
    </InternalElement>
    <InternalElement Name="Running" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/ElementaryStep">
      <ExternalInterface Name="I1" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="I2" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="O1" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
      <ExternalInterface Name="O2" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
      <ExternalInterface Name="O3" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
      <ExternalInterface Name="O4" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
    </InternalElement>
    <InternalElement Name="Aborting" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/MacroStep">
      <ExternalInterface Name="I1" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="O1" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
    </InternalElement>
    <InternalElement Name="Restart" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/MacroStep">
      <ExternalInterface Name="I1" RefBaseClassPath="
          ProcedureBaseICL/StepIn"/>
      <ExternalInterface Name="O1" RefBaseClassPath="
          ProcedureBaseICL/StepOut"/>
```

```
</InternalElement>
<InternalElement Name="StepHold" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/ElementaryStep">
  <ExternalInterface Name="I1" RefBaseClassPath="
      ProcedureBaseICL/StepIn"/>
  <ExternalInterface Name="I2" RefBaseClassPath="
      ProcedureBaseICL/StepIn"/>
  <ExternalInterface Name="O1" RefBaseClassPath="
      ProcedureBaseICL/StepOut"/>
</InternalElement>
<!-- Transitions of the state machine-->
<InternalElement Name="T1" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Idle to Running</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Start</Value>
      </Attribute>
    </InternalElement>
  </InternalElement>
</InternalElement>
<InternalElement Name="T2" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Running to Idle</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Completed</Value>
      </Attribute>
    </InternalElement>
  </InternalElement>
</InternalElement>
<InternalElement Name="T3" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Running to Step Hold</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Lock</Value>
      </Attribute>
    </InternalElement>
  </InternalElement>
</InternalElement>
```

```
<InternalElement Name="T4" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Step Hold to Running</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Unlock</Value>
      </Attribute>
    </InternalElement>
  </InternalElement>
</InternalElement>
<InternalElement Name="T5" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Running to Restart</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Restart</Value>
      </Attribute>
    </InternalElement>
  </InternalElement>
</InternalElement>
<InternalElement Name="T6" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Restart to Step Hold</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Restart completed</Value>
      </Attribute>
    </InternalElement>
  </InternalElement>
</InternalElement>
<InternalElement Name="T7" RefBaseSystemUnitPath="
    ProcedureBaseSUCL/Transition">
  <Description>Running to Aborting</Description>
  <InternalElement Name="Condition" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Condition">
    <InternalElement Name="RR" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/RequestReceived">
      <Attribute Name="value">
        <Value>Abort</Value>
      </Attribute>
    </InternalElement>
```

```
    </InternalElement>
  </InternalElement>
  <InternalElement Name="T8" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Transition">
    <Description>Aborting to Aborted</Description>
    <InternalElement Name="Condition" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/Condition">
      <InternalElement Name="RR" RefBaseSystemUnitPath="
          ProcedureBaseSUCL/RequestReceived">
        <Attribute Name="value">
          <Value>Abort completed</Value>
        </Attribute>
      </InternalElement>
    </InternalElement>
  </InternalElement>
  <InternalElement Name="T9" RefBaseSystemUnitPath="
      ProcedureBaseSUCL/Transition">
    <Description>Aborted to Idle</Description>
    <InternalElement Name="Condition" RefBaseSystemUnitPath="
        ProcedureBaseSUCL/Condition">
      <InternalElement Name="RR" RefBaseSystemUnitPath="
          ProcedureBaseSUCL/RequestReceived">
        <Attribute Name="value">
          <Value>Restart</Value>
        </Attribute>
      </InternalElement>
    </InternalElement>
  </InternalElement>
  <!-- Supported Role-->
  <SupportedRoleClass RefRoleClassPath="ProcedureBaseRCL/
      StateMachine"/>
  <!-- Links between states and transitions-->
  <InternalLink Name="T1in" RefPartnerSideA="Idle:O1"
      RefPartnerSideB="T1:I1"/>
  <InternalLink Name="T1out" RefPartnerSideA="T1:O1"
      RefPartnerSideB="Running:I1"/>
  <InternalLink Name="T2in" RefPartnerSideA="Running:O1"
      RefPartnerSideB="T2:I1"/>
  <InternalLink Name="T2out" RefPartnerSideA="T2:O1"
      RefPartnerSideB="Idle:I1"/>
  <InternalLink Name="T3in" RefPartnerSideA="Running:O2"
      RefPartnerSideB="T3:I1"/>
  <InternalLink Name="T3out" RefPartnerSideA="T3:O1"
      RefPartnerSideB="StepHold:I1"/>
  <InternalLink Name="T4in" RefPartnerSideA="StepHold:O1"
      RefPartnerSideB="T4:I1"/>
  <InternalLink Name="T4out" RefPartnerSideA="T4:O1"
      RefPartnerSideB="Running:I2"/>
  <InternalLink Name="T5in" RefPartnerSideA="Running:O3"
      RefPartnerSideB="T5:I1"/>
```

```
      <InternalLink Name="T5out" RefPartnerSideA="T5:O1"
          RefPartnerSideB="Restart:I1"/>
      <InternalLink Name="T6in" RefPartnerSideA="Restart:O1"
          RefPartnerSideB="T6:I1"/>
      <InternalLink Name="T6out" RefPartnerSideA="T6:O1"
          RefPartnerSideB="StepHold:I2"/>
      <InternalLink Name="T7in" RefPartnerSideA="Running:O4"
          RefPartnerSideB="T7:I1"/>
      <InternalLink Name="T7out" RefPartnerSideA="T7:O1"
          RefPartnerSideB="Aborting:I1"/>
      <InternalLink Name="T8in" RefPartnerSideA="Aborting:O1"
          RefPartnerSideB="T8:I1"/>
      <InternalLink Name="T8out" RefPartnerSideA="T8:O1"
          RefPartnerSideB="Aborted:I1"/>
      <InternalLink Name="T9in" RefPartnerSideA="Aborted:O1"
          RefPartnerSideB="T9:I1"/>
      <InternalLink Name="T9out" RefPartnerSideA="T8:O1"
          RefPartnerSideB="Idle:I2"/>
    </SystemUnitClass>
  </SystemUnitClassLib>
</CAEXFile>
```

Literaturverzeichnis

[1] Abel, D. (1990). *Petri-Netze für Ingenieure: Modellbildung und Analyse diskret gesteuerter Systeme*. Berlin: Springer.

[2] Adolphs, P. und Epple, U. (2015). Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0). URL `https://www.vdi.de/fileadmin/user_upload/VDI-GMA_Statusreport_Referenzarchitekturmodell-Industrie40.pdf`, (besucht am 27.11.2015).

[3] Albrecht, H. (2003). *On Meta-Modeling for Communication in Operational Process Control Engineering*, Nummer 975 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[4] Alt, O. (2012). *Modellbasierte Systementwicklung mit SysML*. München: Hanser.

[5] Årzén, K.-E. (1994). Grafcet for intelligent supervisory control applications. *Automatica 30*(10), S. 1513–1525.

[6] Bauer, N. (2004). *Formale Analyse von Sequential Function Charts*, Nummer 2004,1 in *Schriftenreihe des Lehrstuhls für Anlagensteuerungstechnik der Universität Dortmund*. Aachen: Shaker.

[7] Berners-Lee, T.; Fielding, R. und Masinter, L. (2002). Uniform Resource Identifier (URI): Generic Syntax. URL `https://tools.ietf.org/html/rfc3986`, (besucht am 02.12.2015).

[8] Beutelspacher, A. und Zschiegner, M.-A. (2014). Graphentheorie. In A. Beutelspacher und M.-A. Zschiegner (Hrsgg.), *Diskrete Mathematik für Einsteiger*, S. 175–208. Wiesbaden: Springer Fachmedien Wiesbaden.

[9] Birk, J. und Krauss, M. (2015). Remote Operations in der Prozessautomatisierung: Treiber und Entwicklungen. *atp edition 57*(01-02), S. 60–67.

[10] Böckenhauer, H.-J. und Hromkovič, J. (2013). *Formale Sprachen: Endliche Automaten, Grammatiken, lexikalische und syntaktische Analyse*. Wiesbaden: Springer Vieweg.

[11] Booch, G.; Rumbaugh, J. und Jacobson, I. (2006). *Das UML-Benutzerhandbuch: Aktuell zur Version 2.0*. Programmer's choice. München [u.a.]: Addison-Wesley.

[12] Brandl, D.; Hunkar, P.; Mahnke, W. und Ono, T. (2013). OPC UA and ISA 95: Interoperability for MES by implementing ISA 95 with OPC UA. *atp edition 55*(1-2), S. 64–72.

[13] Bratukhin, A. und Sauter, T. (2010). Bridging the gap between centralized and distributed manufacturing execution planning. In *ETFA 2010: IEEE 15th Conference on Emerging Technologies & Factory Automation*.

[14] Bungartz, H.-J. (2009). *Modellbildung und Simulation: Eine anwendungsorientierte Einführung*. eXamen-press. Berlin: Springer.

[15] Carstensen, K.-U.; Jekat, S. und Klabande, R. (2010). Computerlinguistik - Was ist das? In K.-U. Carstensen (Hrsg.), *Computerlinguistik und Sprachtechnologie*, S. 1–26. Heidelberg: Spektrum Akad. Verl.

[16] Christiansen, L.; Hoernicke, M. und Fay, A. (2014). Modellgestütztes Engineering: Basis für die Automatisierung der Automatisierung. *atp edition 56*(3), S. 18–27.

[17] Deiretsbacher, K.-H. und Mahnke, W. (2014). OPC UA für Industrie 4.0: Sicherer Austausch semantischer Operationen. *atp edition 56*(6), S. 44–51.

[18] Diedrich, C.; Meyer, M.; Evertz, L. und Schäfer, W. (2014). Dienste in der Automatisierungstechnik: Automatisierungsgeräte werden I40-Komponenten. *atp edition 56*(12), S. 24–34.

[19] Diedrich, C. und Riedl, M. (2016). Engineering and integration of automation devices in I40 systems. *at - Automatisierungstechnik 64*(1), S. 41–50.

[20] DIN CLC/TR 62541-1 (2011). OPC Unified Architecture - Teil 1: Übersicht und Konzepte (IEC/TR 62541-1:2010); Deutsche Fassung CLC/TR 62541-1:2010. DIN Deutsches Institut für Normung e. V.

[21] DIN EN 60848 (2014). GRAFCET, Spezifikationssprache für Funktionspläne der Ablaufsteuerung (IEC 60848:2013). DIN Deutsches Institut für Normung e. V.

[22] DIN EN 61131-1 (2004). Speicherprogrammierbare Steuerungen - Teil 1: Allgemeine Informationen (IEC 61131-1:2003). DIN Deutsches Institut für Normung e. V.

[23] DIN EN 61131-3 (2014). Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen (IEC 61131-3:2014). DIN Deutsches Institut für Normung e. V.

[24] DIN EN 61131-3 Beiblatt 1 (2005). Speicherprogrammierbare Steuerungen - Leitlinien für die Anwendung und Implementierung von Programmiersprachen für Speicherprogrammierbare Steuerungen. DIN Deutsches Institut für Normung e. V.

[25] DIN EN 61360-1 (2004). Genormte Datenelementtypen mit Klassifikationsschema für elektrische Bauteile - Teil 1: Definitionen - Regeln und Methoden (IEC 61360-1:2002 + A1:2003); Deutsche Fassung EN 61360-1:2002 + A1:2004. DIN Deutsches Institut für Normung e. V.

[26] DIN EN 61499-1 (2014). Funktionsbausteine für industrielle Leitsysteme - Teil 1: Architektur (IEC 61499-1:2012). DIN Deutsches Institut für Normung e. V.

[27] DIN EN 62264-1 (2014). Integration von Unternehmensführungs- und Leitsystemen - Teil 1: Modelle und Terminologie (IEC 62264-1:2013). DIN Deutsches Institut für Normung e. V.

[28] DIN EN 62264-2 (2014). Integration von Unternehmensführungs- und Leitsystemen - Teil 2: Objekte und Attribute für die Integration von Unternehmensführungs- und Leitsystemen (IEC 62264-2:2013); Englische Fassung EN 62264-2:2013. DIN Deutsches Institut für Normung e. V.

[29] DIN EN 62264-3 (2008). Integration von Unternehmensführungs- und Leitsystemen - Aktivitätsmodelle für das operative Produktionsmanagement (IEC 62264-3:2007). DIN Deutsches Institut für Normung e. V.

[30] DIN EN 62264-4 (2014). Integration von Unternehmensführungs- und Leitsystemen - Teil 4: Attribute des Objektmodells für die Integration des operativen Produktionsmanagements (IEC 65E/384/CD:2013), Text in Englisch. DIN Deutsches Institut für Normung e. V.

[31] DIN EN 62424 (2010). Darstellung von Aufgaben der Prozessleittechnik - Fließbilder und Datenaustausch zwischen EDV-Werkzeugen zur Fließbilderstellung und CAE-Systemen (IEC 62424:2008); Deutsche Fassung EN 62424:2009. DIN Deutsches Institut für Normung e. V.

[32] DIN EN 62714-1 (2015). Datenaustauschformat für Planungsdaten industrieller Automatisierungssysteme - Automation markup language - Teil 1: Architektur und allgemeine Festlegungen (IEC 62714-1:2014); Deutsche Fassung EN 62714-1:2014. DIN Deutsches Institut für Normung e. V.

[33] DIN EN ISO 9001 (2015). Qualitätsmanagementsysteme - Anforderungen (ISO 9001:2015); Deutsche und Englische Fassung EN ISO 9001:2015. DIN Deutsches Institut für Normung e. V.

[34] DIN IEC 60050-351 (2014). Internationales Elektrotechnisches Wörterbuch - Teil 351: Leittechnik (IEC 60050-351:2013). DIN Deutsches Institut für Normung e. V.

[35] DIN IEC 61512-1 (2000). Chargenorientierte Fahrweise - Teil 1: Modelle und Terminologie (IEC 61512-1:1997). DIN Deutsches Institut für Normung e. V.

[36] DIN IEC 61512-2 (2003). Chargenorientierte Fahrweise - Teil 2: Datenstrukturen und Leitfaden für Sprachen (IEC 61512-2:2001). DIN Deutsches Institut für Normung e. V.

[37] DIN IEC 61512-3 (2009). Chargenorientierte Fahrweise - Teil 3: Modelle und Darstellungen von Verfahrens- und Werksrezepten (IEC 61512-3:2008). DIN Deutsches Institut für Normung e. V.

[38] DIN SPEC 40912 (2014). Kernmodelle - Beschreibung und Beispiele. DIN Deutsches Institut für Normung e. V.

[39] Drath, R. (2013). Warum ein Engineering- Weltmodell bisher nicht gelang. *SPS-MAGAZIN 2013*(10). URL `http://www.sps-magazin.de/?inc=artikel/article_show&nr=80045`, (besucht am 27.01.2016).

[40] Drath, R. und Fedai, M. (2004). CAEX - ein neutrales Datenaustauschformat für Anlagendaten - Teil 1. *atp edition 46*(2), S. 52–56.

[41] Drosdowski, G. und Wermke, M. (2006). *DUDEN: Die deutsche Rechtschreibung* (24. Aufl.). Mannheim: Dudenverlag.

[42] Drusinsky, D. und Harel, D. (1994). On the Power of Bounded Concurrency I: Finite Automata. *Journal of the ACM 41*(3), S. 517–539.

[43] Duden Verlag (2015). DUDEN: Produkt, das. URL `http://www.duden.de/rechtschreibung/Produkt`, (besucht am 16.10.2015).

[44] Eckert, K. (2015). *Funktionaler Anwendungsentwurf verteilter Automatisierungssysteme*, Nummer 460 in *Fortschritt-Berichte VDI: Reihe 20, Rechnerunterstützte Verfahren*. Düsseldorf: VDI-Verlag.

[45] Eggersmann, M.; Kausch, B.; Luczak, H.; Marquardt, W.; Schlick, C.; Schneider, N.; Schneider, R. und Theißen, M. (2008). Work Process Models. In M. Nagl und W. Marquardt (Hrsgg.), *Collaborative and distributed chemical engineering*, Nummer 4970 in *Lecture notes in computer science*, S. 126–152. Berlin: Springer.

[46] Eigner, M. (2014). *Modellbasierte virtuelle Produktentwicklung*. Berlin, Heidelberg: Springer Berlin Heidelberg.

[47] Enste, U. (2001). *Generische Entwurfsmuster in der Funktionsbausteintechnik und deren Anwendung in der operativen Prozeßführung*, Nummer 884 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[48] Enste, U. und Müller, J. (2007). *Datenkommunikation in der Prozessindustrie*. München: Oldenbourg.

[49] Epple, U. (2008). Begriffliche Grundlagen der leittechnischen Modellwelt: Teil1: Terminologielehre, Systemmodellierung. *atp edition 50*(04), S. 83–91.

[50] Epple, U. (2011). Merkmale als Grundlage der Interoperabilität technischer Systeme. *at - Automatisierungstechnik 59*(7), S. 440–450.

[51] Epple, U. (2012). Increasing flexibility and functionality in industrial process control: The helpful usage of models, services and cybernetic principles. In *SSD 2012: IEEE 9th International Multi-Conference on Systems, Signals and Devices*.

[52] Epple, U. (2013). Agentenmodelle in der Anlagenautomation. In P. Göhner (Hrsg.), *Agentensysteme in der Automatisierungstechnik*, S. 95–110. Berlin: Springer.

[53] Epple, U. (2015). Industrie 4.0 - Technical Assets: Grundlegende Begriffe, Konzepte, Lebenszyklen und Verwaltung. URL `https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/Statusreport_I40_TechnicalAssets_Begriffe_WEB.pdf`, (besucht am 23.11.2015).

[54] Epple, U. und Polke, M. (1994). Begriffe. In M. Polke (Hrsg.), *Prozeßleittechnik*. München: Oldenbourg.

[55] Epple, U.; Remmel, M. und Drumm, O. (2011). Modellbasiertes Format für RI-Informationen: Verbesserter Datenaustausch für das PLT-Engineering. *atp edition 53*(01-02), S. 62–71.

[56] Erler, D. (2010). *Analyse der Kommunikation zwischen Steuergeräten bei unsicherem Wissen*. Bachelorarbeit, Hochschule Mittweida, Mittweida. URL `monami.hs-mittweida.de/frontdoor/deliver/index/docId/1276/file/Analyse_der_Kommunikation_zwischen_Steuergeraeten_bei_unsicherem_Wissen.pdf`, (besucht am 02.11.2015).

[57] Fay, A.; Diedrich, C.; Thron, M.; Scholz, A.; Schmidt, P. P. und Holm, T. (2015). Wie bekommt Industrie 4.0 Bedeutung? Normen und Standards als semantische Basis. *atp edition 57*(07-08), S. 30–43.

[58] Fay, A.; Schleipen, M. und Mühlhause, M. (2009). Wie kann man den Engineering-Prozess systematisch verbessern? *atp edition 51*(1-2), S. 80–85.

[59] Fay, A.; Vogel-Heuser, B.; Frank, T.; Eckert, K.; Hadlich, T. und Diedrich, C. (2015). Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. *Journal of Systems and Software 101*, S. 221–235.

[60] Fettke, P. (2008). Business Process Modeling Notation. *Wirtschaftsinformatik 50*(6), S. 504–507.

[61] Fimmers, C. (2016). *Erstellung eines XML-Schemas zum Austausch von neutralen Prozedurbeschreibungen (in Bearbeitung)*. Masterarbeit, RWTH Aachen University, Aachen.

[62] Fischer, S. (2015). *Entwurf und Verifikation von Ablaufsteuerungen*, Nummer 2015,1 in *Schriftenreihe des Lehrstuhls für Systemdynamik und Prozessführung*. Aachen: Shaker.

[63] Fittler, H. (2009). Funktionen der Prozessleitebene: Rezeptfahrweise, Führung von Chargenprozessen. In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 63–105. München: Oldenbourg.

[64] Flik, T. (2008). Rechnerorganisation: Informationsdarstellung. In H. Czichos (Hrsg.), *Hütte - Das Ingenieurwissen*, S. 64–75. Berlin and Heidelberg [u.a.]: Springer.

[65] Foltz, C.; Killich, S. und Wolf, M. (2001). K3 User Guide. URL `http://www.iaw.rwth-aachen.de/download/produkte/k3_userguide_2000-11-21.pdf`, (besucht am 22.11.2015).

[66] Freund, J. (2012). *Praxishandbuch BPMN 2.0*. München: Hanser.

[67] Früh, K. F.; Maier, U. und Schaudel, D. (Hrsgg.) (2009). *Handbuch der Prozessautomatisierung: Prozessleittechnik für verfahrenstechnische Anlagen* (4., überarb. Aufl.). München: Oldenbourg.

[68] Gadatsch, A. (2010). *Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis; eine Einführung für Studenten und Praktiker* (6., aktualisierte Aufl.). Studium. Wiesbaden: Vieweg + Teubner.

[69] Gerber, T.; Theorin, A. und Johnsson, C. (2012). Towards a seamless integration between process modeling descriptions at Business and Production levels-work in progress. In *INCOM 2012: 14th IFAC Symposium on Information Control Problems in Manufacturing*.

[70] Grandgirard, E.; Gertosio, C. und Seigneur, J.-M. (2007). Trust Engines to Optimize Semi-Automated Industrial Production Planning. In *ISIE 2007: 2007 IEEE International Symposium on Industrial Electronics*, S. 1814–1819.

[71] Haarmann, H. und Roos, E. (2015). Produktionssysteme der Zukunft - Smart und intuitiv den Standort Deutschland sichern. *atp edition 57*(4), S. 24–32.

[72] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*(3), S. 231–274.

[73] Harel, D. und Rumpe, B. (2000). Modeling Languages: Syntax, Semantics and All That Stuff: Part I: The Basic Stuff. URL `http://www4.in.tum.de/publ/papers/HR00.pdf`, (besucht am 05.03.2016).

[74] Harjunkoski, I.; Nyström, R. und Horch, A. (2009). Integration of scheduling and control—Theory or practice? *Computers & Chemical Engineering 33*(12), S. 1909–1918.

[75] Henßen, R. und Schleipen, M. (2014). Interoperability between OPC UA and AutomationML. *Procedia CIRP 25*, S. 297–304.

[76] Holm, T.; Obst, M.; Fay, A.; Urbas, L.; Albers, T.; Kreft, S. und Hempen, U. (2014). Dezentrale Intelligenz für modulare Automation: Lösungsansätze für die Realisierung modularer Anlagen. *atp edition 56*(11), S. 34–43.

[77] Höme, S.; Grützner, J.; Hadlich, T.; Diedrich, C.; Schnäpp, D.; Arndt, S. und Schnieder, E. (2015). Semantic Industry: Herausforderungen auf dem Weg zur rechnergestützten Informationsverarbeitung der Industrie 4.0. *at - Automatisierungstechnik 63*(2), S. 74–86.

[78] Hopcroft, J. E.; Motwani, R. und Ullman, J. D. (2011). *Einführung in Automatentheorie, formale Sprachen und Berechenbarkeit*. it, Informatik. München: Pearson Studium.

[79] Hubwieser, P. (2004). *Fundamente der Informatik*. München: Oldenbourg.

[80] Hundt, L. (2012). *Durchgängiger Austausch von Daten zur Verhaltensbeschreibung von Automatisierungssystemen*. Berlin: Logos-Verlag.

[81] ISA TR 106 (2013). Procedure Automation for Continuous Process Operations - Models and Terminology. International Society of Automation.

[82] ISO DIS 9000 (2014). Qualitätsmanagementsysteme - Grundlagen und Begriffe (ISO/-DIS 9000:2014). DIN Deutsches Institut für Normung e. V.

[83] ISO/IEC 19510 (2013). Information technology - Object Management Group Business Process Model and Notation. International Organization for Standardization/International Electrotechnical Commission.

[84] Jäger, T.; Fay, A.; Wagner, T. und Lowen, U. (2012). Comparison of engineering results within domain specific languages regarding information contents and intersections. In *SSD 2012: IEEE 9th International Multi-Conference on Systems, Signals and Devices*.

[85] Jasperneite, J. und Niggemann, O. (2012). Systemkomplexität in der Automation beherrschen: Intelligente Assistenzsysteme unterstützen den Menschen. *atp edition 54*(9), S. 36–45.

[86] Jeromin, H. und Epple, U. (2012). Anwendungs- und herstellerneutrales Modell zur Darstellung und Interaktion mit leittechnischen Funktionen. In *AUTOMATION 2012: Der 13. Branchentreff der Mess-und Automatisierungstechnik*. Düsseldorf: VDI-Verlag.

[87] Johnsson, C. (2008). Graphical Languages for Business Processes and Manufacturing Operations. In C. Myung (Hrsg.), *IFAC WC 2008: 17th IFAC World Congress*, IFAC proceedings volumes, S. 13863–13868. Elsevier.

[88] Johnsson, C. und Årzén, K.-E. (1998). Grafchart for recipe-based batch control. *Computers & Chemical Engineering 22*(12), S. 1811–1828.

[89] Johnsson, C. und Årzén, K.-E. (1999). Grafchart and Grafcet: A Comparison between Two Graphical Languages Aimed for Sequential Control Applications. In *IFAC WC 1999: Preprints 14th World Congress of IFAC*, Nummer A, Beijing, P.R. China, S. 19–24.

[90] Junge, H.-D. (1982). *Technische Kybernetik: Grundlagen und Anwendungen Englisch-Deutsch Deutsch-Englisch*. Wiesbaden: Brandstetter Verlag.

[91] Kagermann, H.; Wahlster, W. und Helbig, J. (2013). Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0: Abschlussbericht des Arbeitskreises Industrie 4.0.

[92] Kampert, D. und Epple, U. (2013). A Service Interface for Exchange of Property Information. In *IECON 2013: 39th Annual Conference of the IEEE Industrial Electronics Society*, S. 6922–6927.

[93] Kecher, C. (2011). *UML 2: Das umfassende Handbuch* (4., aktualisierte und erw. Aufl.). Galileo computing. Bonn: Galileo Press.

[94] Keller, G.; Nüttgens, M. und Scheer, A.-W. (1992). Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". *Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlands* (89), S. 1–30. URL `http://www.uni-saarland.de/fileadmin/user_upload/Fachrichtungen/fr13_BWL/professuren/PDF/heft89.pdf`, (besucht am 20.11.2015).

[95] Killich, S.; Luczak, H.; Schlick, C.; Weissenbach, M.; Wiedenmaier, S. und Ziegler, J. (1999). Task modelling for cooperative work. *Behaviour & Information Technology 18*(5), S. 325–338.

[96] Klein, S.; Frey, G. und Litz, L. (2002). A petri net based approach to the development of correct logic controllers. In *INT 2002: Proceedings of the 2nd International Workshop on Integration of Specification Techniques for Applications in Engineering*, S. 116–129.

[97] Kleuker, S. (2013). *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten* (3., korr. und erw. Aufl.). Wiesbaden: Springer Vieweg.

[98] König, R. und Quäck, L. (1988). *Petri-Netze in der Steuerungs- und Digitaltechnik*. München: Oldenbourg.

[99] Koppermann, C. und Möckel, B. (2009). Planen, Errichten und Betreiben automatisierungstechnischer Anlagen: Qualitäts- und Projektmanagement. In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 678–709. München: Oldenbourg.

[100] Lauber, R. und Göhner, P. (1999). *Prozessautomatisierung* (3. Aufl.), Nummer 1/2 in *Prozessautomatisierung*. Berlin: Springer.

[101] Lessen, T. v. (2011). *Geschäftsprozesse automatisieren mit BPEL*. Heidelberg: Dpunkt.verlag.

[102] Libuda, L.; Gutermuth, G. und Heiss, S. (2011). Arbeitsabläufe in der Anlagenplanung optimieren. *atp edition 53*(9), S. 40–51.

[103] Lüder, A.; Hundt, L. und Biffl, S. (2009). On the Suitability of Modeling Approaches For Engineering Distributed Control Systems. In *INDIN 2009: 7th IEEE International Conference on Industrial Informatics*, Piscataway, NJ. IEEE.

[104] Maier, U. (2009). Feldgeräte: Allgemeine Eigenschaften und Kommunikation: Konventionelle Signalübertragung. In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 242–246. München: Oldenbourg.

[105] Maier, U. (2009). Funktionen der Prozessleitebene: Übersicht über prozessnahe Funktionen. In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 56–62. München: Oldenbourg.

[106] Maier, U. (2009). Geräte der Prozessleitebene: Automatisierungsstrukturen. In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 180–190. München: Oldenbourg.

[107] Maier, U. (2009). Geräte der Prozessleitebene: Speicherprogrammierbare Steuerungen (SPS). In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 205–215. München: Oldenbourg.

[108] Mealy, G. H. (1955). A method for synthesizing sequential circuits. *The Bell System Technical Journal 34*(5), S. 1045–1079.

[109] Mersch, H. (2016). *Deterministische, dynamische Systemstrukturen in der Automatisierungstechnik*, Nummer 1245 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[110] Mersch, H.; Behnen, D.; Schmitz, D.; Epple, U.; Brecher, C. und Jarke, M. (2011). Gemeinsamkeiten und Unterschiede der Prozess- und Fertigungstechnik. *at - Automatisierungstechnik 59*(1).

[111] Mersch, H. und Epple, U. (2011). Requirements on distribution management for service-oriented automation systems. In *ETFA 2011: IEEE 16th Conference on Emerging Technologies & Factory Automation Factory Automation*, S. 1–8.

[112] Mertens, M. (2012). *Verwaltung und Verarbeitung merkmalbasierter Informationen: vom Metamodell zur technischen Realisierung*, Nummer 1207 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[113] Meyer, D. (2002). *Objektverwaltungskonzept für die operative Prozeßleittechnik*, Nummer 940 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[114] Moore, E. F. (1956). Gedanken-Experiments on Sequential Machines. In C. Shannon und J. McCarthy (Hrsgg.), *Automata Studies*, S. 129–153. Princeton, NJ: Princeton University Press.

[115] Mühlhause, M. (2012). *Konzept zur durchgängigen Nutzung von Engineeringmodellen der Automation*. Berlin: Logos-Verlag.

[116] Münnemann, A. (2005). *Infrastrukturmodell zur Integration expliziter Verhaltensbeschreibungen in die operative Prozessleittechnik*, Nummer 1068 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[117] N. N. (2008). Extensible Markup Language (XML) 1.0 -W3C Recommendation. URL `https://www.w3.org/TR/xml/`, (besucht am 03.03.2016).

[118] N. N. (2012). OWL 2 Web Ontology Language. URL `https://www.w3.org/TR/owl2-overview/`, (besucht am 03.03.2016).

[119] N. N. (2013). BatchML. URL `https://services.mesa.org/ResourceLibrary/ShowResource/4cf7f45a-7b4f-4e3a-b600-43fe7b691f49`, (besucht am 03.03.2016).

[120] N. N. (2014). Industrie 4.0 Statusreport: Gegenstände, Entitäten, Komponenten. URL `https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/VDI_Industrie_4.0_Komponenten_2014.pdf`, (besucht am 09.12.2015).

[121] N. N. (2014). Industrie 4.0 Statusreport: Wertschöpfungsketten. URL `https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/sk_dateien/VDI_Industrie_4.0_Wertschoepfungsketten_2014.pdf`, (besucht am 18.12.2015).

[122] N. N. (2014). JGrafchart Manual. URL `http://www.control.lth.se/grafchart/doc/JGrafchart-2.6.1-doc/`, (besucht am 11.11.2015).

[123] N. N. (2015). OMG Systems Modeling Language. URL `http://www.omg.org/spec/SysML/1.4/PDF/`, (besucht am 03.03.2016).

[124] N. N. (2015). OPC Unified Architecture: Interoperabilität für Industrie 4.0 und das Internet der Dinge. URL `https://opcfoundation.org/wp-content/uploads/2015/04/OPC-UA-Interoperability-For-Industrie4-and-IoT-DE1.pdf`, (besucht am 03.08.2015).

[125] N. N. (2015). Umsetzungsstrategie Industrie 4.0: Ergebnisbericht der Plattform Industrie 4.0. URL https://www.bmwi.de/BMWi/Redaktion/PDF/I/industrie-40-verbaendeplattform-bericht,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf, (besucht am 25.11.2015).

[126] N. N. (2016). Industrie 4.0 Statusreport: Durchgängiges Engineering in Industrie 4.0-Wertschöpfungsketten. URL https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/6032_PUB_TW_GMA_Statusreport_Durchgaengiges_Engineering_Internet.pdf, (besucht am 14.03.2016).

[127] Nahrstedt, H. (2009). *C++ für Ingenieure*. Studium. Wiesbaden: Vieweg + Teubner.

[128] NE 139 (2012). Informationsschnittstellen in der Prozessautomatisierung. NAMUR.

[129] NE 150 (2012). Standardisierte NAMUR-Schnittstelle zum Austausch von Engineering-Daten zwischen CAE-System und PCS-Engineering-Werkzeugen. NAMUR.

[130] NE 160 (2016). Ein Referenzmodell für allgemeine Prozedurbeschreibungen (in Begutachtung). NAMUR.

[131] Nielen, A. (2014). *Systematik für die leistungs- und zuverlässigkeitsorientierte Modellierung von Arbeitsprozessen mit kontrollflussorientierten Notationssystemen*, Nummer 15 in *Industrial engineering and ergonomics*. Aachen: Shaker.

[132] OASIS Web Services Business Process Execution Language (WSBPEL) TC (2007). OASIS Web Services Business Process Execution Language (WSBPEL) TC: OASIS Standard. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, (besucht am 20.10.2015).

[133] Obst, M.; Hahn, A. und Urbas, L. (2014). Package-Unit-Integration in der Prozessindustrie: Was fehlt für Plug-and-produce? *atp edition 56*(1-2), S. 56–65.

[134] Obst, M.; Holm, T.; Urbas, L.; Fay, A.; Kreft, S.; Hempen, U. und Albers, T. (2015). Beschreibung von Prozessmodulen: Ein weiterer Schritt zur Umsetzung der NE 148. *atp edition 57*(01-02), S. 48–59.

[135] Obst, M.; Urbas, L.; Runde, S.; Wolf, G.; Maurmaier, M. und Rachut, H. (2014). Automation der Integration von Package Units in Leitsysteme. In *Automation 2014: Der 15. Branchentreff der Mess- und Automatisierungstechnik*.

[136] Ollinger, L.; Zühlke, D.; Theorin, A. und Johnsson, C. (2013). A reference architecture for service-oriented control procedures and its implementation with SysML and Grafchart. In *ETFA 2013: IEEE 18th Conference on Emerging Technologies & Factory Automation*, S. 1–8.

[137] Olsson, R. (2002). *Exception Handling in Recipe-Based Batch Control*. Licentiate Thesis, Lund University. URL https://lup.lub.lu.se/search/publication/1044342, (besucht am 18.10.2015).

[138] Paelke, V.; Röcker, C.; Koch, N.; Flatt, H. und Büttner, S. (2015). User interfaces for cyber-physical systems: Expanding the designer's toolbox. *at - Automatisierungstechnik 63*(10), S. 833–843.

[139] PLCopen XML (2009). XML Formats for IEC 61131-3. PLCopen Technical Committee 6. URL http://www.plcopen.org/pages/tc6_xml/, (besucht am 29.02.2016).

[140] Polke, M. (1994). Einleitung. In M. Polke (Hrsg.), *Prozeßleittechnik*, S. 11–19. München: Oldenbourg.

[141] Priese, L. und Wimmel, H. (2008). *Petri-Netze*. eXamen-press. Berlin: Springer.

[142] Reisig, W. (1986). *Petrinetze*. Studienreihe Informatik. Berlin: Springer.

[143] Runde, S. und Drumm, O. (2012). Service-orientierte Methode zum Datenaustausch zwischen Engineering-Werkzeugen: Neuer Ansatz - alte Probleme? In U. Jumar, E. Schnieder, und C. Diedrich (Hrsgg.), *EKA 2012: Entwurf komplexer Automatisierungssysteme: Beschreibungsmittel Methoden Wekzeuge und Anwendugen*, S. 253–266.

[144] Schleipen, M.; Münnemann, A. und Sauer, O. (2011). Interoperabilität von Manufacturing Execution Systems (MES). *at - Automatisierungstechnik 59*(7), S. 413–424.

[145] Schlick, C. M.; Bruder, R. und Luczak, H. (2010). *Arbeitswissenschaft*. Springer Berlin Heidelberg.

[146] Schmitz, S.; Schluetter, M. und Epple, U. (2009). Automation of Automation — Definition, components and challenges. In *ETFA 2009: IEEE 14th Conference on Emerging Technologies & Factory Automation*.

[147] Schnieder, E. (1992). *Petrinetze in der Automatisierungstechnik*. Oldenbourg Wissenschaftsverlag.

[148] Schor, W. und Große, N. (2008). Hierarchien und Symbolik in der Produktionstechnik. *atp edition 50*(11), S. 66–74.

[149] Schuler, H. (2009). Höhere Ebenen: Informationsverbund und MES: Prozessleittechnik im Informationsverbund des Unternehmens. In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 16–33. München: Oldenbourg.

[150] Schüller, A.; Elger, J. und Epple, U. (2015). A Cybernetic Approach to Create Real-Time Optimized Supply Chains. In *ACC 2015: 2015 American Control Conference*.

[151] Schüller, A. und Epple, U. (2014). Ein einheitliches Prozedurbeschreibungsmodell als Basis der domänenübergreifenden Verständigung. In *EKA 2014: Entwurf komplexer Automatisierungssysteme*.

[152] Schüller, A. und Epple, U. (2015). Ein Referenzmodell zur Prozedurbeschreibung - Eine Basis für Industrie 4.0. *at - Automatisierungstechnik 63*(2), S. 87–98.

[153] Schüller, A. und Epple, U. (2015). Prozedurbasierte Assistenzsysteme: Fallbasiertes Schließen mithilfe einer Prozedurdatenbank. *atp edition 57*(7/8), S. 62–69.

[154] Schüller, A.; Epple, U.; Elger, J.; Müller-Martin, A. und Löwen, U. (2014). Geschäftsprozesse und technische Prozesse - Ein Ebenenmodell zur Integration. *at - Automatisierungstechnik 62*(9), S. 665–675.

[155] Schüller, A.; Scholz, A.; Tauchnitz, T.; Drath, R. und Scherwietes, T. (2015). Speed-Standardisierung am Beispiel der PLT-Stelle: Datenaustausch mit dem NAMUR-Datencontainer. *atp edition 57*(1-2), S. 36–46.

[156] Schumacher, F. (2013). *Automatische Generierung von IEC 61131-3 Steuerungscode aus einer GRAFCET-Spezifikation*, Nummer 450 in *Fortschritt-Berichte VDI: Reihe 20, Rechnerunterstützte Verfahren*. Düsseldorf: VDI-Verlag.

[157] Schumacher, F. und Fay, A. (2013). Transforming time constraints of a GRAFCET graph into a suitable Petri net formalism. In *ICIT 2013: 2013 IEEE International Conference on Industrial Technology*, S. 210–218.

[158] Schumacher, F. und Fay, A. (2014). Petrinetzmodell für die Formalisierung von GRAFCET-Spezifikationen. *at - Automatisierungstechnik 62*(6), S. 385–393.

[159] Schumann, A. und Reinhold, D. (2009). Höhere Ebenen: Informationsverbund und MES: Manufacturing Execution Systems (MES). In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 34–54. München: Oldenbourg.

[160] Seidlmeier, H. (2015). *Prozessmodellierung mit ARIS®: Eine beispielorientierte Einführung für Studium und Praxis in ARIS 9* (4., aktualisierte Aufl.). Wiesbaden: Springer Fachmedien.

[161] Shobrys, D. E. und White, D. C. (2002). Planning, scheduling and control systems: why cannot they work together. *Computers & Chemical Engineering 26*(2), S. 149–160.

[162] Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Wien: Springer Wien. URL `https://ia800904.us.archive.org/22/items/Stachowiak1973AllgemeineModelltheorie/Stachowiak%20%281973%29:%20Allgemeine%20Modelltheorie.pdf`, (besucht am 23.10.2015).

[163] Steusloff, H. (1994). Die leittechnische Anlage und ihre Elemente: Informationslogistik: Funktionsstrukturen und Informationsfluß in Produktionsunternehmen. In M. Polke (Hrsg.), *Prozeßleittechnik*, S. 536–538. München: Oldenbourg.

[164] Tauchnitz, T. und Maier, U. (2009). Geräte der Prozessleitebene: Prozessleitsysteme (PLS). In K. F. Früh, U. Maier, und D. Schaudel (Hrsgg.), *Handbuch der Prozessautomatisierung*, S. 191–204. München: Oldenbourg.

[165] Theorin, A. (2014). *A Sequential Control Language for Industrial Automation*. Ph.D. Thesis, Lund University, Lund.

[166] Theorin, A.; Hagsund, J. und Johnsson, C. (2014). Service Orchestration with OPC UA in a Graphical Control Language. In *ETFA 2014: 2014 IEEE Emerging Technology and Factory Automation*.

[167] Theorin, A.; Ollinger, L. und Johnsson, C. (2012). Service-oriented process control with grafchart and the devices profile for web services. In *INCOM 2012: 14th IFAC Symposium on Information Control Problems in Manufacturing*.

[168] Theurich, S.; Schüller, A.; Wollschläger, M. und Epple, U. (2014). Dienstbasierte Prüfung von CAEX-Exporten mit standardisierten Bibliotheken. In *Automation 2014: Der 15. Branchentreff der Mess- und Automatisierungstechnik*.

[169] Thramboulidis, K. (2015). An open distributed architecture for flexible hybrid assembly systems: A model-driven engineering approach. *The International Journal of Advanced Manufacturing Technology*.

[170] Tran, T. D. (2014). *Ermittlung nichtfunktionaler Anforderungen an Assistenzsysteme in der Prozessautomation*. Bachelorarbeit, RWTH Aachen University, Aachen.

[171] Uhlig, R. und Bruns, M. (1995). *Automatisierung von Chargenprozessen: Mit 20 Tabellen*. München: Oldenbourg.

[172] Ulrich, A.; Güttel, K. und Fay, A. (2009). Durchgängige Prozesssicht in unterschiedlichen Domänen: Methoden und Werkzeug zum Einsatz der formalisierten Prozessbeschreibung. *at - Automatisierungstechnik 57*(2), S. 80–92.

[173] van der Aalst, W. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin and Heidelberg: Springer.

[174] VDI/VDE 2653-1 (2010). Agentensysteme in der Automatisierungstechnik - Grundlagen. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[175] VDI/VDE 3681 (2005). Einordnung und Bewertung von Beschreibungmitteln aus der Automatisierungstechnik. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[176] VDI/VDE 3682 (2005). Formalisierte Prozessbeschreibung. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[177] VDI/VDE 3690-1 (2013). XML in der Automation - Klassifikation ausgewählter Anwendungen. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[178] VDI/VDE 3690-2 (2012). XML in der Automation - Überführung fachlicher Modelle nach XML. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[179] VDI/VDE 3695-3 (2010). Engineering von Anlagen - Evaluieren und optimieren des Engineerings - Themenfeld Methoden. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[180] VDI/VDE 3696-2 (1995). Herstellerneutrale Konfigurierung von Prozeßleitsystemen - Standard-Funktionsbausteine. VDI/VDE Gesellschaft Mess- und Automatisierungstechnik.

[181] Venkatasubramanian, V.; Rengaswamy, R.; Yin, K. und Kavuri, S. N. (2003). A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers & Chemical Engineering 27*(3), S. 293–311.

[182] Vogel-Heuser, B.; Diedrich, C. und Broy, M. (2013). Anforderungen an CPS aus Sicht der Automatisierungstechnik. *at - Automatisierungstechnik 61*(10), S. 669–676.

[183] Vogel-Heuser, B.; Diedrich, C.; Fay, A.; Jeschke, S.; Kowalewski, S.; Wollschlaeger, M. und Göhner, P. (2014). Challenges for Software Engineering in Automation. *Journal of Software Engineering and Applications 7*(05), S. 440–451.

[184] Wagner, C. und Epple, U. (2015). Sprechende Kommandos als Grundlage moderner Prozessführungsschnittstellen. In *Automation 2015: Der 16. Branchentreff der Mess- und Automatisierungstechnik*.

[185] Wagner, C.; Kampert, D.; Schüller, A.; Palm, F.; Grüner, S. und Epple, U. (2016). Model Based Synthesis of Automation Functionality. *at - Automatisierungstechnik 64*(3).

[186] Wagner, T. und Löwen, U. (2010). Modellierung: Grundlage für integriertes Engineering. In *Automation 2010: Der 11. Branchentreff der Mess- und Automatisierungstechnik*, Düsseldorf. VDI-Verlag.

[187] Wandke, H. und Wetzenstein-Ollenschläger, E. (2003). Assistenzsysteme: Woher und wohin? In *Tagungsband UP03*. Stuttgart: Fraunhofer Verlag.

[188] Weerawarana, S. (2006). *Web services platform architecture*. Upper Saddle River, NJ: Prentice Hall PTR.

[189] Wehmeier, S. (1993). *Oxford wordpower dictionary*. Oxford: Oxford University Press.

[190] Weilkiens, T.; Weiss, C. und Grass, A. (2010). *Basiswissen Geschäftsprozessmanagement: Aus- und Weiterbildung zum OMG Certified Expert in Business Process Management (OCEB) Fundamental Level* (1. Aufl.). Heidelberg: Dpunkt.verlag.

[191] Wellenreuther, G. und Zastrow, D. (2009). *Automatisieren mit SPS - Theorie und Praxis* (4., überarb. und erw. Aufl.). Wiesbaden: Vieweg + Teubner.

[192] Witsch, D. (2013). *Modellgetriebene Entwicklung von Steuerungssoftware auf Basis der UML unter Berücksichtigung der domänenspezifischen Anforderungen des Maschinen- und Anlagenbaus* (1. Aufl.). Göttingen: Sierke.

[193] Witsch, D.; Ricken, M.; Kormann, B. und Vogel-Heuser, B. (2010). PLC-Statecharts: An Approach to Integrate UML-Statecharts in Open-Loop Control Engineering. In *INDIN 2010: 8th IEEE International Conference on Industrial Informatics*, S. 915–920.

[194] Witsch, D. und Vogel-Heuser, B. (2011). PLC-Statecharts: An Approach to Integrate UML-Statecharts in Open-Loop Control Engineering: Aspects on Behavioral Semantics and Model-Checking. In B. Sergio (Hrsg.), *IFAC WC 2011: 18th IFAC World Congress*, S. 7866–7872. Elsevier.

[195] Witsch, M. (2014). *Funktionale Spezifikation von Manufacturing Execution Systems im Spannungsfeld zwischen IT, Geschäftsprozess und Produktion* (1. Aufl.). Göttingen: Sierke.

[196] Yu, L. (2016). *A Reference Model for the Integration of Agent Orientation in the Operative Environment of Automation Systems*, Nummer 1248 in *Fortschritt-Berichte VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI-Verlag.

[197] Yu, L.; Grüner, S. und Epple, U. (2013). An Engineerable Procedure Description Method for Industrial Automation. In *ETFA 2013: IEEE 18th Conference on Emerging Technologies & Factory Automation*.

[198] Yu, L.; Quirós, G. und Epple, U. (2011). An Assistance System Approach for Flexible Product Transport Operations in Process Plants. In B. Sergio (Hrsg.), *IFAC WC 2011: 18th IFAC World Congress*, S. 7304–7309. Elsevier.

[199] Yu, L.; Schüller, A. und Epple, U. (2014). On the engineering design for systematic integration of agent-orientation in industrial automation. *ISA transactions 53*(5), S. 1404–1409.

[200] Zacher, S. und Reuter, M. (2014). *Regelungstechnik für Ingenieure*. Lehrbuch. Wiesbaden: Springer Vieweg.

[201] Zur Muehlen, M. und Recker, J. (2008). How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In Z. Bellahsène (Hrsg.), *Advanced Information Systems Engineering: CAiSE*, Nummer 5074 in *Lecture notes in computer science*, S. 465–479. Berlin: Springer.

Online-Buchshop für Ingenieure

VDI nachrichten **BUCHSHOP**

Online-Shops

Fachliteratur und mehr - jetzt bequem online recherchieren & bestellen unter: www.vdi-nachrichten.com/Der-Shop-im-Ueberblick

Täglich aktualisiert: Neuerscheinungen VDI-Schriftenreihen

Im Buchshop von vdi-nachrichten.com finden Ingenieure und Techniker ein speziell auf sie zugeschnittenes, umfassendes Literaturangebot.

Mit der komfortablen Schnellsuche werden Sie in den VDI-Schriftenreihen und im Verzeichnis lieferbarer Bücher unter 1.000.000 Titeln garantiert fündig.

Im Buchshop stehen für Sie bereit:

VDI-Berichte und die Reihe **Kunststofftechnik:**

Berichte nationaler und internationaler technischer Fachtagungen der VDI-Fachgliederungen

Fortschritt-Berichte VDI:

Dissertationen, Habilitationen und Forschungsberichte aus sämtlichen ingenieurwissenschaftlichen Fachrichtungen

Newsletter „Neuerscheinungen":

Kostenfreie Infos zu aktuellen Titeln der VDI-Schriftenreihen bequem per E-Mail

Autoren-Service:

Umfassende Betreuung bei der Veröffentlichung Ihrer Arbeit in der Reihe Fortschritt-Berichte VDI

Buch- und Medien-Service:

Beschaffung aller am Markt verfügbaren Zeitschriften, Zeitungen, Fortsetzungsreihen, Handbücher, Technische Regelwerke, elektronische Medien und vieles mehr – einzeln oder im Abo und mit weltweitem Lieferservice

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-525408-6