Felix Oestersötebier

Modellbasierter Entwurf intelligenter mechatronischer Systeme mithilfe semantischer Technologien

Model-based Design of Intelligent Mechatronic Systems Using Semantic Web Technologies

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie detaillierte bibliografische Daten sind im Internet über `http://dnb.ddb.de` abrufbar

Band 378 der Verlagsschriftenreihe des Heinz Nixdorf Instituts

ISSN (Print): 2195-5239
ISSN (Online): 2365-4422
ISBN: 978-3-942647-97-7

Satz und Gestaltung: Felix Oestersötebier

Hersteller: readbox unipress in der readbox publishing GmbH
Münster

Printed in Germany

Modellbasierter Entwurf intelligenter mechatronischer Systeme mithilfe semantischer Technologien

zur Erlangung des akademischen Grades eines
DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)
der Fakultät Maschinenbau
der Universität Paderborn

genehmigte
DISSERTATION

von
Dipl.-Ing. Felix Oestersötebier
aus Gütersloh

Tag des Kolloquiums: 18. Dezember 2017
Referent: Prof. Dr.-Ing. habil. Ansgar Trächtler
Korreferent: Prof. Dr.-Ing. Jürgen Gausemeier

Vorwort

Diese Dissertation entstand in meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Regelungstechnik und Mechatronik des Heinz Nixdorf Instituts der Universität Paderborn. Sie ist geprägt durch die Mitarbeit in dem EU-Forschungsprojekt „Entwurfstechnik intelligente Mechatronik“ (ENTIME) und dem Innovationsprojekt „Intelligenter und optimierter Teigknetprozess“ (InoTeK) im Rahmen des Spitzenclusters it's OWL.

Für die Möglichkeit, diese Arbeit an seinem Lehrstuhl durchführen zu können, die inhaltliche und moralische Unterstützung sowie für das entgegengebrachte Vertrauen, möchte ich mich an dieser Stelle herzlich bei meinem Doktorvater, Herrn Prof. Dr.-Ing. habil. Ansgar Trächtler bedanken. Weiterhin gilt mein Dank Herrn Prof. Dr.-Ing. Jürgen Gausemeier für die Übernahme des Korreferats. Bei Prof. Dr.-Ing. Roman Dumitrescu und Prof. Dr. rer. nat. Thomas Tröster bedanke ich mich für das Mitwirken in der Prüfungskommission.

Außerdem möchte ich die Chance nutzen, mich bei allen nun ehemaligen Kollegen des Lehrstuhls für das angenehme Arbeitsklima und die gute Zusammenarbeit zu bedanken. Die Zeit mit Euch, sowohl auf als auch neben der Arbeit, die Doktorandenseminare und Lehrstuhlausflüge, werde ich niemals vergessen. Besonders hervorheben möchte ich in diesem Zusammenhang meinen Dank an Matthias Lochbichler, Daniel Kruse und Dr.-Ing. Viktor Just für die intensiven fachlichen Diskussionen und Ratschläge sowie die sorgfältige Durchsicht des Manuskripts. Den ehemaligen Studenten und späteren Kollegen Phillip Traphöner und Peng Wang sowie allen anderen „meiner“ Bachelor-, Studien- und Masterarbeiter und studentischen Hilfskräfte danke ich für den Unterstützung, die sie mit ihren jeweiligen Arbeiten geleistet haben.

Zu guter Letzt gilt besonderer Dank meinen Eltern und meiner Familie, auf deren Rückhalt und Verständnis ich mich immer verlassen konnte und kann.

Paderborn, im Februar 2018 *Felix Oestersötebier*

Vorveröffentlichungen

[1] Tichy, Matthias; Oestersötebier, Felix; Schierbaum, Thomas: Entwurfstechnik Intelligente Mechatronik. In: *Internationales Forum Mechatronik*, 21.–22. Sep. 2011, S. 427–440

[2] Oestersötebier, Felix; Just, Viktor; Trächtler, Ansgar; Bauer, Frank; Dziwok, Stefan: Model-Based Design of Mechatronic Systems by Means of Semantic Web Ontologies and Reusable Solution Elements. In: *Proceedings of the ASME 2012 11th Biennial Conference on Engineering Systems Design and Analysis*, Nantes, Frankreich, 2.–4. Jul. 2012, S. 647–656

[3] Oestersötebier, Felix; Dziwok, Stefan; Bauer, Frank; Trächtler, Ansgar; Schäfer, Wilhelm; Gausemeier, Jürgen: Unterstützung des mechatronischen Entwurfs durch die effektive Suche nach Lösungselementen mithilfe von semantischen Technologien. In: *Tag des Systems Engineerings*, Nov. 2012 Gesellschaft für Systems Engineering e.V. (GfSE), Carl Hanser Verlag, München, S. 207–216

[4] Bauer, Frank; Gausemeier, Jürgen; Köchling, Daniel; Oestersötebier, Felix: Simulative Absicherung mechatronischer Systeme in der frühen Phase der Produktentstehung. In: *Tag des Systems Engineerings*, Nov. 2012 Gesellschaft für Systems Engineering e.V. (GfSE), Carl Hanser Verlag, München, S. 197–206

[5] Bauer, Frank; Gausemeier, Jürgen; Köchling, Daniel; Oestersötebier, Felix: Approach for an Early Validation of Mechatronic Systems using Idealized Simulation Models within the Conceptual Design. In: *Smart Product Engineering - Proceedings of the 23rd CIRP Design Conference*, Mrz. 2013, Springer Berlin/Heidelberg, S. 273–282

[6] Pohlmann, Uwe; Trsek, Henning; Dürkop, Lars; Dziwok, Stefan; Oestersötebier, Felix: Application of an Intelligent Network Architecture on a Cooperative Cyber-Physical System: An Experience Report. In: *19th IEEE Conference on Emerging Technologies and Factory Automation (Work-in-Progress) EFTA2014*, Barcelona, Spanien, Sep. 2014

[7] Oestersötebier, Felix; Wang, Peng; Trächtler, Ansgar: A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces. In: *Proceedings of the 10th International Modelica Conference*, Lund, Schweden, Mrz. 2014, S. 929–937

[8] Gausemeier, Jürgen (Hrsg.); Trächtler, Ansgar (Hrsg.); Schäfer, Wilhelm (Hrsg.): *Semantische Technologien im Entwurf mechatronischer Systeme: Effektiver Austausch von Lösungswissen in Branchenwertschöpfungsketten.* ISBN 978-3446436305, Jun. 2014, Carl Hanser Verlag München
(Insbesondere Abschnitte: 2.1.2, 2.2.2, 3.4.1, 3.6.1, 3.7.1, 3.7.2, 3.9.5)

[9] Lochbichler, Matthias; Oestersötebier, Felix; Trächtler, Ansgar: Dynamic Behavior Models and Their Modeling Depth in the Design Process of Mechatronic Systems. In: *Proceedings of the ASME 2014 International Mechanical Engineering Congress & Exposition IMECE 2014*, Montreal, Kanada, Nov. 2014

[10] Lankeit, Christopher; Lochbichler, Matthias; Oestersötebier, Felix; Trächtler, Ansgar; Landwehr, Martin: From Goals to Systems via Technical Requirements and Physical Models. In: *Proceedings of the first IEEE International Symposium on Systems Engineering IEEE ISSE 2015*, Rom, Italien, Sept. 2015

[11] Abrishamchian, Farisoroosh; Oestersötebier, Felix; Trächtler, Ansgar: Feature Model Approach for Managing Variability of Dynamic Behavior Models in Mechatronic Systems. In: *Proceedings of the ASME 2015 International Mechanical Engineering Congress & Exposition IMECE 2015*, Houston, Texas, Nov. 2015

[12] Oestersötebier, Felix; Traphöner, Phillip; Reinhart, Felix; Wessels, Sebastian; Trächtler, Ansgar: Design and Implementation of Intelligent Control Software for a Dough Kneader. In: *Proceedings of the 3rd International Conference on System-Integrated Intelligence*, Paderborn, 13. - 15. Jun. 2016

[13] Oestersötebier, Felix; Abrishamchian, Farisoroosh; Lankeit, Christopher; Just, Viktor; Trächtler, Ansgar: Approach for an Integrated Model-Based System Design of Intelligent Dynamic Systems Using Solution and System Knowledge. In: *Proceedings of the 3rd International Conference on System-Integrated Intelligence*, Paderborn, 13.–15. Jun. 2016

Zusammenfassung

In dieser Arbeit wird eine interdisziplinäre, modellbasierte Methodik für den Entwurf intelligenter, mechatronischer Systeme vorgestellt. Mithilfe dieser Methodik sollen der Aufwand, die Komplexität und die Fehleranfälligkeit bei der Entwicklung reduziert werden, indem Lösungs- und Systemwissen semantisch aufbereitet und genutzt werden. Unter dem Begriff Lösungswissen werden wiederverwendbare Elemente, Produkte oder Muster, aber z. B. auch speziell für den Zweck der zielgerichteten Wiederverwendung erstellte Dynamikmodelle zur modellbasierten Analyse und Synthese verstanden. Systemwissen bezeichnet systemspezifische Zusammenhänge, Strukturinformationen und Eigenschaften. Beides wird mithilfe semantischer Technologien so aufbereitet, dass unterschiedliche Klassifikationen, Terminologien, Abstraktionsgrade und zugrundeliegende Denkweisen überwunden werden können und damit die nahtlose Integration in den interdisziplinären Entwurfsprozess gelingt. Andererseits wird es auf diese Weise ermöglicht, implizit vorhandenes Wissen herzuleiten, indem die modellierten logischen Zusammenhänge und Regeln ausgewertet werden. Sowohl die Modellierung von wiederverwendbaren Dynamikmodellen sowie von Lösungs- und Systemwissen als auch dessen Nutzung/Integration wird anhand von Beispielen gezeigt. Darüber hinaus wird die Umsetzbarkeit einer Werkzeugunterstützung prototypisch evaluiert.

Abstract

In this thesis an interdisciplinary and model-based method for the design of intelligent mechatronic systems is introduced. Thus, it is intended to reduce effort, complexity and error-proneness of the development. The approach is to use semantically enriched solution/system knowledge. The term solution knowledge stands for reusable elements, products or patterns as well as e. g. dynamic behavior models that are particularly build-up and provided for this purpose. Those models are used for the analysis and synthesis of the dynamic system behavior. The notion of system knowledge is to subsume system-specific interrelation, structure and characteristics. Both types of knowledge are modeled by means of semantic technologies. Thus, different classifications, terminologies, degrees of abstraction and ways of thinking can be overcome. On the other hand, implicitly available knowledge can be inferred from the logical relations and rules modeled. The modeling of both the reusable dynamic behavior models and the solution/system knowledge as well as the usage/integration is shown by means of several examples. Furthermore, the feasibility of a tool-support is evaluated by prototypes.

Inhaltsverzeichnis

Abkürzungsverzeichnis

A-Box	Assertional Box
ABS	Anti-Blockiersystem
AMS	Autonomes mechatronisches System
BCS	Body Coordinate System
CAD	Computer-Aided Design
CAE	Computer-Aided Engineering
CFD	Computational Fluid Dynamics
CONSENS	CONceptual design Specification technique for the ENgineering of complex Systems
CPS	Cyber-Physical Systems
CWA	Closed World Assumption
DAE	Differential Algebraic Equation
DLL	Dynamic Link Library
ENTIME	Entwurfstechnik Intelligente Mechatronik
ESP	Elektronisches Stabilitätsprogramm
FEM	Finite-Elemente-Methode
FMI	Functional Mock-Up Interface
FMU	Functional Mock-Up Unit
HiL	Hardware-in-the-Loop
INCOSE	International Council on Systems Engineering
InoTeK	Intelligenter und optimierter Teig-Knetprozess
IRI	Internationalized Resource Identifier
Kfz	Kraftfahrzeug
KIEF	Knowledge Intensive Engineering Framework
KMU	kleine und mittlere Unternehmen

LCS	Local Coordinate System
LE	Lösungselement
LM	Lösungsmuster
MFM	Mechatronisches Funktionsmodul
MiL	Model-in-the-Loop
MKS	Mehrkörpersystem
MSL	Modelica Standard Library
ODE	Ordinary Differential Equation
OMG	Object Management Group
OSLC	Open Services for Lifecycle Collaboration
OWA	Open World Assumption
OWL	Web Ontology Language
PLIB	Parts Library
PLM	Produktlebenszyklusmanagement
PWM	Pulsweitenmodulation
QUDT	Quantities, Units, Dimensions and Data Types Ontologies
RCP	Rapid Control Prototyping
RDF	Ressource Description Framework
SE	Systems Engineering
SiL	Software-in-the-Loop
SPARQL	SPARQL Protocol And RDF Query Language
SPIN	SPARQL Inferencing Notation
SQL	Structured Query Language
SQWRL	Semantic Query-Enhanced Web Rule Language
STEP	Standard for the exchange of product model data
SWRL	Semantic Web Rule Language
SysML	Systems Modeling Language
T-Box	Terminological Box
TCP	Tool-Center-Point

TGG	Triple Graph Grammar
UML	Unified Modeling Language
UNA	Unique Name Assumption
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VMS	Vernetztes mechatronisches System
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Symbolverzeichnis

Γ	Modellierungstiefe eines Dynamikmodells
γ	Gewichtungsfaktor zur Bestimmung der Gesamtmodellierungstiefe eines zusammengesetzten Modells
$\gamma_{1..6}$	Konstanten zur stetig differenzierbaren Approximation der Stribeck-Reibungskurve
Γ^*	gewünschte Zielmodellierungstiefe eines Dynamikmodells
$\theta_{1,2,3}$	Gelenkwinkel der Delta-Kinematik
K	Komplexität eines Dynamikmodells
μ	Reibungsbeiwert
$\mu_{s,k}$	Gleit- bzw. Haftreibungskoeffizienten
ω_N	Nenndrehzahl eines Servoantriebs
ω_{rel}	Relativdrehzahl von Bottich und Spirale
ω_S	Drehfrequenz der Knetspirale
$a_{1,2}$	Schwerpunkte der Kontaktpartner bei Kontaktmodellen
$b_{1,2}$	(Potentielle) Kontaktpunkte bei Kontaktmodellen
b_{WS}	Breite des Arbeitsraums einer Delta-Kinematik
c	Kontaktsteifigkeit
c_d	Elastizität des Teiges
C_p	Spezifische Wärmekapazität des Teiges
d	Kontaktdämpfung
d_d	Viskosität/Dämpfung des Teiges
$\underline{e}$	Einheitsvektor
E_s	in den Teig eingebrachte spezifische Energie
$\underline{F}_K$	Kontaktkraft bei Kontaktmodellen
F_N	Normalkraft bei Kontaktmodellen
f_{PWM}	PWM-Frequenz eines Servoantriebs
F_T	Tangentialkraft bei Kontaktmodellen
h_d	Wärmeübergangskoeffizient für den Wärmeaustausch von Teig und Umgebung
I_{Max}	maximaler Strom eines Servoantriebs
J_{Load}	(Äquivalentes) Lastträgheitsmoment eines Servoantriebs

$\underline{J}(\underline{q})$	Jakobi-Matrix
J_R	Rotorträgheitsmoment eines Servoantriebs
J_w	Gütekriterium für die Geometrie einer Delta-Kinematik auf Basis der Manipulierbarkeit
K_M	Drehmomentkonstante eines Servoantriebs
k_v	Koeffizient für viskose Reibung
$L_{A,B}$	Längen von Ober- und Unterarm der Delta-Kinematik
L_i	Induktivität des Stromkreises eines Servoantriebs
l_{WS}	Länge des Arbeitsraums einer Delta-Kinematik
m	Eindringungsexponent
$m_{A,B}$	Massen von Ober- und Unterarm der Delta-Kinematik
M_d	Widerstandsmomemt des Teiges
m_d	Teigmasse
$M_{d\varphi}$	niederfrequentes Widerstandsmomemt des Teiges
$M_{d\omega}$	hochfrequentes Widerstandsmomemt des Teiges
m_E	Endeffektormasse einer Delta-Kinematik
M_{loss}	Verlustmoment an der Knetspirale
M_{Max}	maximales Drehmoment eines Servoantriebs
$M_{mot,S}$	Antriebsmoment der Knetspirale
M_N	Nenndrehmoment eines Servoantriebs
M_R	konstantes Reibmoment eines Servoantriebs
n	Steifigkeitsexponent
n_{min}	Anzahl der Minimalzustände bzw. Pole der Matrixübertragungsfunktion
$\underline{P}$	Projektionsmatrix
p	Eindringtiefe (Penetration) bei Kontaktmodellen
$P_{d,loss}$	Thermische Verlustleistung des Teiges
$P_{d,mech,(f)}$	in den Teig eingebrachte (gefilterte) mechanische Leistung
$P_{d,therm}$	in den Teig eingebrachte thermische Leistung
R	Radius der Schlägerplatte der Delta-Roboter
$\underline{r}_{1,2}$	Richtungsvektor
r	Radius des Spielballs der Delta-Roboter
$R_{A,B}$	Radien der Koppelpunkte von Grundplatte und Endeffektorplattform der Delta-Kinematik
r_E	Endeffektorradius einer Delta-Kinematik
R_i	Widerstand des Stromkreises eines Servoantriebs

$\underline{r}_{TCP}$	Ortsvektor der TCP-Position der Delta-Kinematik/ Endeffektorkoordinaten
S_d	Teigoberfläche
T_d	Teigtemperatur
T_{env}	Umgebungstemperatur
T_i	Zeitkonstante des Stromregelkreises eines Servoantriebs
T_T	Zeitkonstante für die Verzögerung der Umsetzung von mechanischer in thermische Teigenergie
T_w	Ausgangstemperatur des Wassers
$\ddot{u}$	Getriebeübersetzungsverhältnis
U_{Net}	Versorgungsspannung eines Servoantriebs
$v_{\epsilon,1,2}$	Grenzgeschwindigkeit für Misch- bzw. Gleitreibung
v_{rel}	Relative Geschwindigkeit der Kontaktpunkte
w	Manipulierbarkeit einer Roboter-Kinematik

1 Einleitung

Betrachtet man die Entwicklung technischer Systeme in den letzten zwei Jahrzehnten, so stellt man fest, dass der Anteil klassischer, rein mechanischer Systeme deutlich zurückgegangen ist. Stattdessen kann man einen massiven Anstieg von solchen technischen Systemen beobachten, die sich dadurch auszeichnen, dass Bestandteile aus mehreren Technikdisziplinen zusammenwirken. Diese Verknüpfung wird durch den Begriff „Mechatronik“ zum Ausdruck gebracht. Das klassische Beispiel für ein mechatronisches System ist das Kraftfahrzeug (Kfz). Hatten Kfz in den 1970er Jahren neben dem mechanischen Aufbau nur ein einziges (Motor-) Steuergerät, so hielten in den nachfolgenden Jahrzehnten zunehmend weitere Elektronik- und Softwarekomponenten Einzug (siehe Tabelle 1-1). Diese übernehmen sowohl Sicherheits- als auch Komfortfunktionen oder zum Teil auch beides gleichzeitig: ESP und ABS tragen zum Beispiel nachweislich signifikant zur Verbesserung der Unfallstatistik bei [GPL+08, Mey10]. Das Fahrlicht schaltet sich heute nicht nur automatisch an, unterscheidet zwischen Tagfahrlicht und Abblendlicht, sondern es leuchtet auch die Kurve und sogar die Gegenfahrbahn aus, ohne den entgegenkommenden Verkehr zu blenden. Fenster und Schiebedach müssen nicht mehr manuell „aufgekurbelt“ werden und schließen sich z. T. sogar automatisch, wenn ein Überschlag droht[1].

Tabelle 1-1: Zuwachs an Komplexität von Elektrik/Elektronik-Architekturen am Beispiel der Mercedes-Benz E-Klasse [SKS+11]

Baureihe	Baujahr (Limousine)	Anzahl Fahrzeugbusse	Steuergeräte	Kommunikationssignale
W124	1984–1995	1	7	unter 100
W210	1995–2002	3	30	ca. 200
W211	2002–2009	5	52	ca. 4100
W212	2009–2016	9	ca. 70	ca. 6000

1.1 Definition und Aufbau mechatronischer Systeme

Verstand man unter „Mechatronik“ zu Beginn der 1980er Jahre noch vor allem die Verbindung von Mechanik und Elektronik, so verwendet man den Begriff heute deutlich allgemeiner für eben jenes synergetische Zusammenwirken von verschiedenen technischen Disziplinen. Neben dem faktisch integralen Bestandteil einer

[1] Siehe z. B. Mercedes Pre-Safe-System

softwarebasierten Informationsverarbeitung sind teilweise auch hydraulische, thermodynamische oder pneumatische Teilsysteme anzutreffen. Durch die geschickte räumliche und/oder funktionale Integration erreicht man flexible, anpassungsfähige, robuste und zunehmend vernetzte, kurz intelligente Systeme, die einen Mehrwert gegenüber der Summe ihrer Bestandteile hervorbringen. Ausgehend von früheren Formulierungen ([HT96, IFA05] u. a.) kommt ISERMANN daher zur folgenden, erweiterten Definition:

> *„Mechatronische Systeme entstehen durch simultanes Entwerfen und die Integration von folgenden Komponenten oder Prozessen:*
>
> - *Mechanische und mit ihnen gekoppelte Komponenten/Prozesse*
> - *Elektronische Komponenten/Prozesse*
> - *Informationstechnik (einschließlich Automatisierungstechnik)*
>
> *Die Integration erfolgt durch die Komponenten (Hardware) und durch die informationsverarbeitenden Funktionen (Software). Ziel ist dabei, eine optimale Lösung zu finden zwischen der mechanischen Struktur, Sensor- und Aktor-Implementierung, automatischer digitaler Informationsverarbeitung und Regelung. Zusätzlich werden synergetische Effekte geschaffen, die erweiterte Funktionen und innovative Lösungen ergeben.“* [Ise08, S. 18]

Bild 1-1 zeigt den hierarchischen Aufbau mechatronischer Systeme (vgl. [LKS00], [HSN+97] sowie [VDI04]). Auf der untersten Ebene – dem `Mechatronischen Funktionsmodul (MFM)` – finden sich die vier wiederkehrenden Grundbausteine eines mechatronischen Systems, die über `Energie-`, `Stoff-` und `Signalflüsse` miteinander in Verbindung stehen (vgl. [PBF+13, VDI04]). Das `Grundsystem` stellt eine passive, zumeist mechanische Anordnung dar, deren momentaner Zustand durch einen `Sensor` erfasst wird. Dieser leitet die Information an die `Informationsverarbeitung` weiter, welche den geeigneten Eingriff zur Beeinflussung des Grundsystems errechnet und mittels eines `Aktors` umsetzt. Mechatronische Funktionsmodule können ihrerseits Bestandteile eines `Autonomen mechatronischen Systems (AMS)` sein und darin Aufgaben übernehmen. Sie werden in der Regel von einer übergeordneten Informationsverarbeitung koordiniert. Kommunizieren AMS miteinander, so spricht man von `Vernetzten mechatronischen Systemen (VMS)` oder auch *Cyber-Physical Systems (CPS)* bzw. *Systems-of-Systems*. Auch hier gibt es vielfach eine übergeordnete Kontrollinstanz. Zum Beispiel bilden Motor, Getriebe etc. im autonomen mechatronischen System Kraftfahrzeug je ein mechatronisches Funktionsmodul mit Aktor, Sensor und eigener Informationsverarbeitung. Kfz kommunizieren jedoch heute zunehmend auch miteinander [CCC11] und stellen somit vernetzte mechatronische Systeme dar.

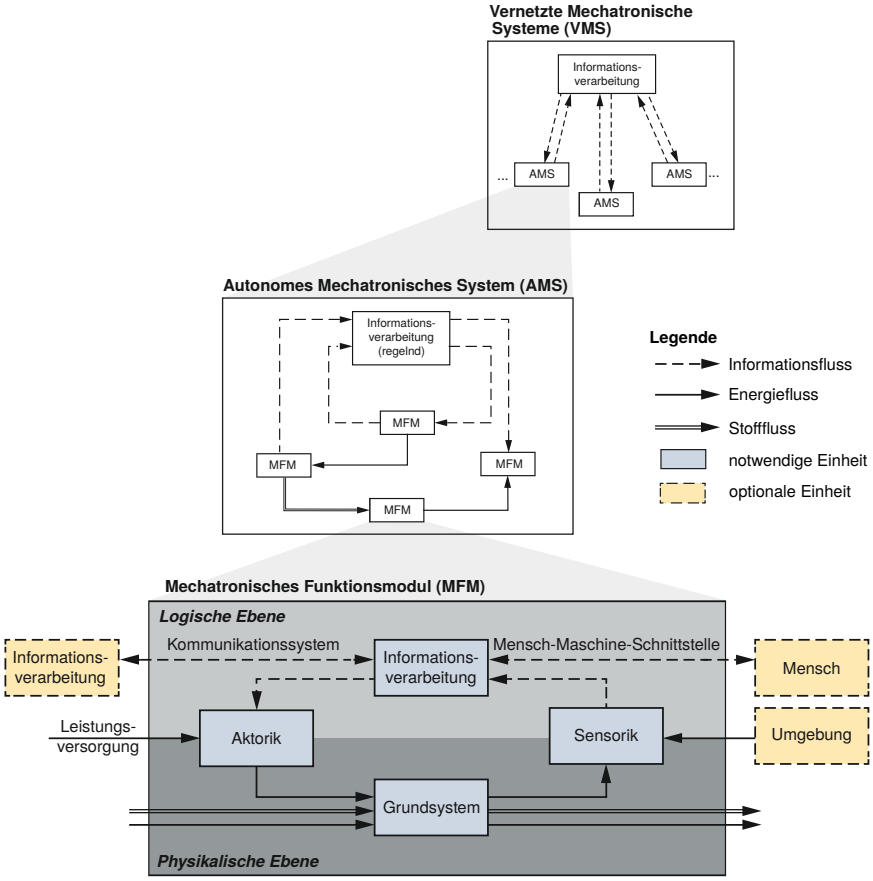


Bild 1-1: Hierarchischer Aufbau komplexer mechatronischer Systeme, vgl. [GTS14, S. 27–29]

1.2 Motivation und Zielsetzung

Je höher die Hierarchieebene komplexer mechatronischer Systeme, desto größer wird der Softwareanteil des Systems und desto komplexer wird seine Entwicklung. Die Vielzahl der beteiligten Fachdisziplinen und Fachexperten steigert den Abstimmungs- und Koordinationsbedarf zusätzlich. Die Entwicklung von intelligenten technischen Systemen ist daher fehleranfällig. Dies belegt u. a. die hohe Zahl der Kfz-Rückrufe (vgl. Bild 1-2), die in den letzten Jahren durch große

Anstrengungen der Hersteller zwar leicht rückläufig, dennoch im Vergleich zu den 1990er Jahren deutlich gestiegen ist [Mey10].

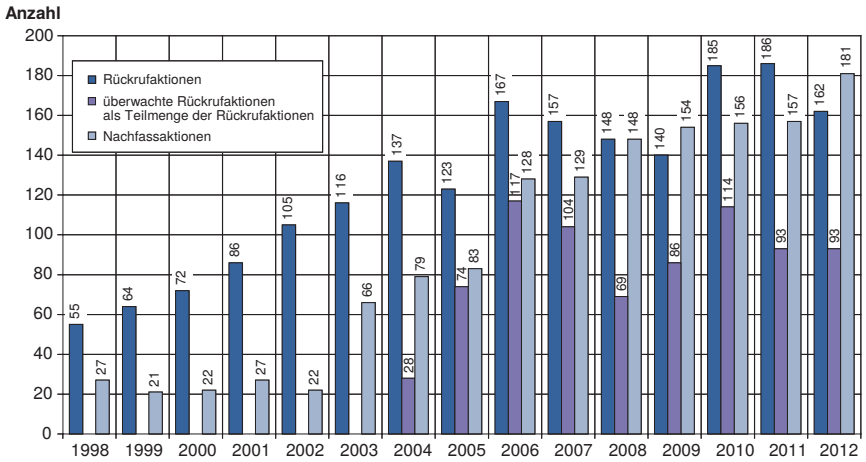


Bild 1-2: Anzahl der Kfz-Rückrufaktionen[2] in Deutschland von 1998 bis 2012 [KBA12, S. 58]

Die größten Fehlerpotentiale bilden laut Ehrlenspiel [EM13, S. 142] folgende Faktoren:

- zu wenig Zusammenarbeit,
- zu wenig Kommunikation und
- zu wenig Koordination.

Die immerwährende Forderung nach kürzeren Entwicklungszeiten und *Simultaneous Engineering*[3] verschärft die Situation noch zusätzlich. Iterationen bleiben während der Entwicklung jedoch weiterhin notwendig und unvermeidlich [EM13]. Das Resultat all dessen ist, dass in vielen Entwicklungen heute eine große Vielzahl von Entwicklungsdokumenten und Artefakten existiert, die nicht konsistent ist. Inkonsistenzen besitzen logischerweise ein großes Fehlerpotential. Es geht also darum, sie möglichst gar nicht erst entstehen zu lassen, damit Fehlerpotentiale zu eliminieren und Fehler zu vermeiden. Gleichzeitig muss sichergestellt werden, dass

[2] Nachfassaktionen betreffen Fahrzeuge, deren Halter sich nicht bei ihrer Werkstatt gemeldet haben, um den Mangel beseitigen zu lassen, nachdem sie zum ersten Mal darüber informiert wurden [KBA12].

[3] Mit dem Begriff Simultaneous Engineering verbindet man das Bestreben, die Entwicklung in den einzelnen Disziplinen zu parallelisieren.

Fehler, die nicht vermieden werden konnten, möglichst frühzeitig entdeckt werden, denn die Kosten für ihre Beseitigung steigen exponentiell mit dem Fortschreiten des Zeitpunkts der Entdeckung. Clark und Fujimoto formulierten hierzu die *rule of ten* [CF91], nach der sich die Kosten von einem Entwicklungsschritt zum nächsten jeweils um den Faktor 10 vergrößern (siehe Bild 1-3).

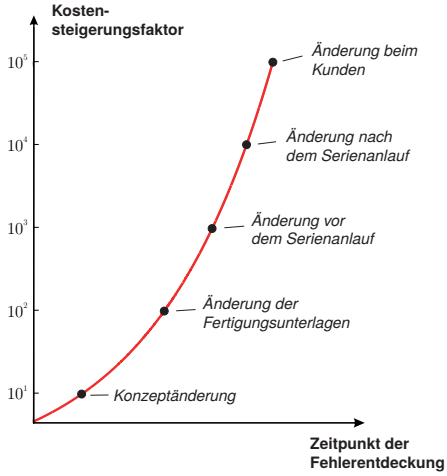


Bild 1-3: Kostensteigerung für die Fehlerbehebung: „rule of ten“ vgl. [CF91], [EM13, S. 143]

In dieser Arbeit sollen zwei Strategien adressiert werden, um der Komplexität und Fehleranfälligkeit bei der Entwicklung intelligenter technischer Systeme zu begegnen:

- Übergreifendes modellbasiertes Entwickeln,
- Auffinden und Wiederverwenden von Lösungswissen.

Beide werden an dieser Stelle nun kurz erläutert.

1) Übergreifendes modellbasiertes Entwickeln: Rechnermodelle spielen bereits heute eine Schlüsselrolle in vielen Bereichen des Entwicklungsprozesses. Vor allem in der Ausarbeitung eines neu zu entwickelnden Systems kommen häufig verschiedenartige, auf den speziellen Anwendungszweck der jeweiligen Fachdisziplin zugeschnittene Modelle zum Einsatz. Die modellbasierte Entwicklung ist jedoch vielfach nicht durchgehend. Gerade in den frühen Phasen der Entwicklung birgt die konsequente Nutzung von Modellen im Allgemeinen – und von idealisierten Simulationsmodellen im Speziellen – noch großes Potential, um Fehler und überflüssige Iterationen frühzeitig zu verhindern. Systemtests können vor der realen

Umsetzung im Rechner durchgeführt und Arbeitsschritte parallelisiert werden. Darüber hinaus kann die Funktionsfähigkeit des Systems mithilfe detaillierter Simulationsmodelle vor der Inbetriebnahme genauer überprüft werden, sodass sich große Sicherheitsfaktoren sowie nachträgliche Anpassungen vermeiden lassen. Besonderes Augenmerk soll hier auf die Simulation des dynamischen (Bewegungs-) Verhaltens gelegt werden, da dieses im besonderen Maße Interdisziplinarität erfordert und die Erfüllung der Anforderungen in diesem Bereich oft kritisch ist. Mithilfe von Dynamikmodellen können die systemspezifischen Anforderungen sowohl für die nachfolgenden Entwicklungsschritte als auch für die Auswahl von wiederverwendbaren Teillösungen deutlich präzisiert werden. Die Modelle müssen hierfür allerdings den speziellen Bedürfnissen des jeweiligen Entwicklungsschritts gerecht werden.

Darüber hinaus können Systemmodelle, d. h. Modelle, die das Zusammenwirken der einzelnen Systembestandteile und Entwicklungsartefakte wiedergeben, helfen die oben genannte Fehlerquellen zum Versiegen zu bringen. Sie sollen Systemwissen explizit verfügbar machen und Abhängigkeiten aufzeigen, sodass zum einen in der Konzeptphase ein gemeinsames Verständnis des zu entwickelnden Systems geschaffen und zum anderen ein „Auseinanderdriften“ der Ausarbeitungen in den einzelnen Fachdisziplinen vermieden wird. In einigen wenigen Bereichen und Branchen tragen Systemmodelle so bereits heute zu Komplexitätsmanagement, Kommunikation und Koordination bei. Von einer flächendeckenden Durchdringung dieser Methoden in der Praxis kann allerdings bisher keinesfalls gesprochen werden [GDS+13, TT14]. Jedoch wächst mit der Komplexität des Systems auch die des Systemmodells, wodurch das Modell trotz vielfacher Abstraktion schnell unübersichtlich und manuell schwer verständlich und analysierbar wird. Speziell ausgebildete Systemingenieure müssen es aufbauen und pflegen. In der konkreten Umsetzung findet man sowohl integrative Ansätze (z. B. [RPC+12]), die versuchen, disziplinspezifische Teilmodelle in einem großen Modell zu integrieren, als auch föderative Ansätze, die darauf abzielen, die Modelle und Entwicklungsdokumente in den spezialisierten Tools der einzelnen Disziplinen zu verknüpfen und konsistent zu halten (z. B. [GSG+09]).

In dieser Arbeit soll eine föderative Methode zur Erstellung eines Gesamtsystemmodells unter Nutzung semantischer Technologien vorgestellt werden. Sie ermöglicht es den Entwicklern, nahezu ausschließlich in ihren bekannten und spezialisierten Tools zu arbeiten und gleichzeitig das semantische Systemmodell (auch Systemwissensmodell genannt) mit Inhalt zu füllen. Dieses Wissensmodell beinhaltet das gesammelte Systemwissen und ermöglicht es, je nach Frage-/Problemstellung nur die jeweils relevanten Bestandteile abzufragen und darzustellen. Des Weiteren kann durch Schlussfolgerungen implizit vorliegendes Systemwissen explizit gemacht und eine etwaige logische Inkonsistenz aufgedeckt werden. Hierdurch soll

der Zielkonflikt zwischen abgestimmter, übergreifender Entwicklungsarbeit auf der einen und innovativer, effizienter sowie spezialisierter Entwicklung in kleinen, unabhängigen Arbeitsgruppen auf der anderen Seite entschärft werden.

2) Auffinden und Wiederverwenden von Lösungswissen: Sowohl der Forderung nach kürzeren Entwicklungszeiten als auch der Fehlerträchtigkeit komplexer Entwicklungen lässt sich durch effizientes Wissensmanagement und Wiederverwendung bewährter Lösungen begegnen. Gerade in kleinen und mittleren Unternehmen (KMU) wird oftmals ein konstruktionsgetriebener Entwurf für neue Systeme verfolgt, bei dem die Entwickler zwar grundsätzlich Lösungswissen für die benötigten Teillösungen in Form von sog. Lösungselementen nutzen, die sie von Zulieferern beziehen, jedoch häufig immer wieder die gleichen, bekannten Produkte verwenden [GSA+11]. Die eingesetzten Lösungselemente sind zur Erfüllung der Aufgabe allerdings nicht immer optimal, da die systemspezifischen Anforderungen an die Teillösungen zum einen häufig nicht genau bekannt sind, zum anderen die manuelle Suche nach dem optimalen Lösungselement (LE) extrem aufwendig und in der Praxis nicht wirtschaftlich ist [ODB+12]. Auch hieraus resultieren vielfach unnötig große Sicherheitsfaktoren.

Das Ziel dieser Arbeit ist es daher, die Suche nach passenden Lösungselementen in den modellbasierten Entwicklungsprozess zu integrieren und durch den Einsatz semantischer Technologien zu unterstützen. Dazu wird die Idee des „Semantic Web“ nach Berners-Lee aufgegriffen [BF99]. Seine Vision sieht eine Erweiterung der im Internet verfügbaren Inhalte vor, sodass die Informationen sowohl für den Menschen als auch für den Computer „verständlich“ sind. Bezogen auf den Anwendungsfall ist die Grundidee die Erweiterung der bereitgestellten Informationen über Lösungselemente im Internet durch den Anbieter, damit die rechnerunterstützte Suche und Auswahl passender Teillösungen ermöglicht wird. Hierbei sind sowohl diejenigen Lösungselemente gemeint, die extern bezogen werden, als auch solche die unternehmensintern verfügbar sind. Als Ausgangsbasis für die Suche und Auswahl dienen die oben erwähnten idealisierten Dynamikmodelle des Systems, welche die Zielgrößen liefern. In diesem Zusammenhang stellen Dynamikmodelle extrem wertvolles Lösungswissen dar, das wiederverwendet werden kann und soll. Denn dadurch kann der Modellierungsaufwand bei der Erstellung eines Dynamikmodells des Gesamtsystems signifikant verringert werden. Dieser Aufwand ist ein wesentlicher Hinderungsgrund für eine breitere Nutzung modellbasierter Methoden, vor allem bei KMU.

Zusammenfassend sollen demzufolge Methoden entstehen, die den durchgängig modellbasierten Entwurf mechatronischer Systeme – bei Neu- wie auch bei Anpassungsentwicklungen – unterstützen (vgl. Bild 1-4). Die Unterstützung erfolgt mithilfe semantischer Technologien und hilft sowohl bei der Ausarbeitung entlang des Entwurfsprozesses (hier „Unterstützung durch Lösungswissen“ genannt) als

auch quer zum Entwurfsprozess bei der Koordination der verschiedenen Disziplinen (hier „Unterstützung durch Systemwissen“ genannt). Durch die Verknüpfung und Abstimmung dieser beiden Ansätze wird weiterhin ein nahtloser Informationsrückfluss für spätere Entwicklungen im Sinne einer Wissenskonservierung ermöglicht. Als Beispiel für fachdisziplinspezifische Modelle werden im Folgenden vornehmlich Dynamikmodelle verwendet, da gerade die Dynamikanforderungen an technische Systeme vielfach maßgeblich sind. Die Aussagen und Ergebnisse sind jedoch im Grundsatz immer auch auf andere, beteiligte Disziplinen übertragbar.

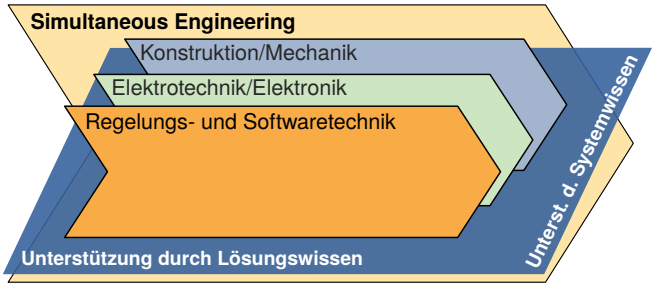


Bild 1-4: Unterstützung des Entwurfsprozesses durch Lösungs- und Systemwissen

1.3 Aufbau der Arbeit

Nachfolgend wird zunächst ein Überblick über den Stand der Wissenschaft und Technik zu den Bereichen „Modellbasierter Entwurf“ und „Model-Based Systems Engineering“ gegeben (Kapitel 2). Im Abschnitt 2.4 wird insbesondere auf semantische Technologien und wissensbasierte Systeme eingegangen, da diese Themen im Kontext des Entwurfs intelligenter mechatronischer Systeme in der Regel wenig Verwendung finden und dementsprechend nicht als bekannt vorausgesetzt werden können. Zum Abschluss des Kapitels werden verwandte Arbeiten diskutiert (Abschnitt 2.5) und der Handlungsbedarf hergeleitet.

Kapitel 3 stellt anschließend das Entwurfsvorgehen anhand zweier Beispielsysteme dar. Bei der Entwicklung kooperierender Delta-Roboter handelt es sich um eine Neuentwicklung, die sich durch eine hohe Dynamik und eine anspruchsvolle Informationsverarbeitung sowie eine enge Verzahnung der Disziplinen Mechanik, Regelungstechnik und Softwaretechnik auszeichnet (Abschnitt 3.1). Abschnitt 3.2 erläutert das Vorgehen bei der Anpassungsentwicklung eines intelligenten Teig-

kneters. Das Ziel besteht darin, ein bereits bestehendes Knetsystem zu optimieren[4], vor allem im Hinblick auf seine Intelligenz.

Das Kapitel 4 befasst sich mit der Aufbereitung von Modellen zum Zweck der Wiederverwendung im Entwurf mechatronischer Systeme. Als Beispiel dienen hierzu Dynamikmodelle. Nach einer Definition des Begriffs *Modellierungstiefe* und der Abgrenzung zum Begriff *Modellkomplexität* in Abschnitt 4.1 wird das Vorgehen zur Aufbereitung von Dynamikmodellen in der geeigneten Modellierungstiefe thematisiert (siehe Abschnitt 4.2). Anschließend werden ausführlich mehrere Beispiele für Dynamikmodelle in der Konzipierungsphase (Abschnitt 4.3) und in der Ausarbeitungsphase (Abschnitt 4.4) erläutert.

Im Kapitel 5 wird aufgezeigt, wie Lösungswissen aufbereitet werden muss (siehe Abschnitt 5.1), um es in den Entwurfsprozess zu integrieren (vgl. Abschnitt 5.2). Dazu wird eine Lösungselementspezifikation eingeführt. Abschnitt 5.2.3 beschreibt die prototypische Umsetzung des erarbeiteten Konzepts.

Die Unterstützung des Entwurfsprozesses mithilfe eines semantischen Systemmodells (Systemwissen) wird in Kapitel 6 aufgegriffen. Hier wird analog ebenfalls zunächst die Aufbereitung und Modellierung von Systemwissen beschrieben (Abschnitt 6.1 und 6.2). Im Abschnitt 6.3 wird zunächst anhand eines Beispiels erläutert, wie der Informationsaustausch zwischen den verschiedenen disziplinspezifischen Modellen und Entwicklungsartefakten geschehen kann, ehe auch hier die prototypische Werkzeugunterstützung thematisiert wird.

Zum Abschluss dieser Arbeit wird eine Zusammenfassung gegeben und ein Fazit gezogen (siehe Kapitel 7). Darüber hinaus folgt ein Ausblick auf mögliche, zukünftige Forschungsthemen.

[4] Optimierung ist in diesem Zusammenhang als gezielte Verbesserung und nicht im Sinne einer mathematischen Optimierung, z. B. durch Minimierung einer Zielfunktion, zu verstehen.

2 Grundlagen und Stand der Wissenschaft und Technik

Dieses Kapitel soll einen Überblick über einige im Kontext dieser Arbeit wichtige Grundlagen sowie den Stand der Wissenschaft und Technik geben. Den Kern bilden die beiden Themengebiete „Modellbasierter Entwurf mechatronischer Systeme“ (vgl. Abschnitt 2.2) bzw. „Model-based Systems Engineering“ (vgl. Abschnitt 2.3) und „Semantische Technologien“ (vgl. Abschnitt 2.4). Anschließend folgt eine Übersicht verwandter Arbeiten (Abschnitt 2.5). Zum Abschluss wird ein Fazit gezogen und der Handlungsbedarf abgeleitet (vgl. Abschnitt 2.6). Um Verwirrungen und Missverständnissen im weiteren Verlauf dieser Arbeit vorzubeugen, sollen an dieser Stelle zunächst der Begriff des „Modells“ allgemein definiert und verschiedene Ausprägungen voneinander abgegrenzt werden.

2.1 Modelle und Meta-Modelle

Der Modellbegriff ist gerade in technischen Disziplinen heute omnipräsent. Dabei haben die an der Entwicklung eines mechatronischen Systems beteiligten Akteure – wie Systemingenieure, Mechatroniker, Regelungstechniker, Softwareentwickler oder Konstrukteure – höchst unterschiedliche und z. T. widersprüchliche Vorstellungen, wenn sie den Begriff „Modell“ verwenden. Auch im weiteren Verlauf dieser Arbeit kommt der Begriff in diversen Zusammenhängen vor. Daher wird er im Folgenden kurz allgemein definiert. Zudem werden Unterscheidungskriterien genannt. Die Darstellungen orientieren sich dabei an denen aus [GTS14].

2.1.1 Der Modellbegriff im Allgemeinen

Abstrahiert man von jeglichen konkreten Vorstellungen, so kann man unter einem Modell allgemein ein materielles oder immaterielles Abbild eines realen Originals verstehen. Dieses Abbild dient einem speziellen Zweck und soll helfen, bestimmte Fragestellungen zu beantworten. Dabei werden zahlreiche Vereinfachungen und Abstraktionen verwendet, um die Komplexität zu verringern. Mithilfe dieses Abbilds können dann Experimente durchgeführt werden, die so entweder gar nicht oder nur mit großem Aufwand direkt am Original umsetzbar wären, z. B. weil das Original noch gar nicht existiert [Fri04]. Bild 2-1 veranschaulicht dies. Ein Modell dient damit u. a. zur Problemlösung, zur Strukturierung und zum besseren Verständnis des Originals.

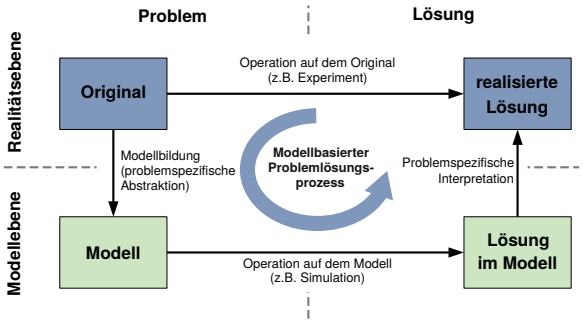


Bild 2-1: Problemlösung mithilfe von Modellen, vgl. [GTS14, S. 30]

Nach Stachowiak [Sta73, S. 128–133] wird der „allgemeine Modellbegriff“ wie folgt definiert:

> *Ein Modell ist eine beschränkte Repräsentation (Abstraktions-, Verkürzungsmerkmal) von Entitäten[1] und Beziehungen der wirklichen Welt (Abbildungsmerkmal) mit einer eindeutigen Korrespondenz für ein festes Zeitintervall und für einen vorher festgelegten Zweck (pragmatisches Merkmal).*

Die Schwierigkeit bei der Erstellung eines Modells besteht darin, von genau den Eigenschaften zu abstrahieren, die für den Zweck nicht relevant sind. Im Gegensatz dazu ist für die Übertragbarkeit der Modellergebnisse (Interpretation) entscheidend, dass diejenigen Eigenschaften des Originals, die für die Bearbeitung der jeweiligen Aufgabenstellung von Bedeutung sind, möglichst gut beibehalten werden (siehe u. a. [Wal95]). Hieraus resultiert die vielfach zitierte Forderung (vgl. z. B. [Neu90, S. 16]): „Ein Modell soll so einfach wie möglich und so genau wie nötig sein.“ Anders ausgedrückt, muss es das Ziel sein, dass alle Operationen und Experimente, die auf dem Modell ausgeführt werden, auch am Original zu gleichen oder ausreichend ähnlichen[2] Ergebnissen führen. Denn nur dann sind die am Modell gewonnen Erkenntnisse auf die Realität übertragbar.

Neben der Reduktion der Komplexität ergeben sich durch die Nutzung von Modellen vielfach weitere Vorteile gegenüber dem Original. Modelle können schneller verfügbar, flexibler, leichter, kleiner und besser transportabel sein. Experimente am Modell sind reproduzierbar, in der Regel ungefährlich, besser kontrollierbar und meist kostengünstiger. Dem Nutzen von Modellen steht der zusätzliche Auf-

[1] Eine Entität bezeichnet in der Philosophie (Ontologie) einen real existierenden Untersuchungsgegenstand [Blu13].

[2] Ein Verfahren zur Überführung von Ergebnissen bei lediglich „ähnlichen“ Modellen ist zum Beispiel das *Buckingham'sche* Π-*Theorem* [Buc14].

wand zur Erstellung des Modells gegenüber. Durch die genannten Vorteile wird er jedoch vielfach an anderer Stelle mehr als kompensiert.

2.1.2 Abgrenzung technischer Modelle

Kastens und Kleine Büning [KK08] unterscheiden folgende Aspekte, die durch ein Modell unter anderem betrachtet werden können.

- Die **Struktur** eines Systems beschreibt, aus welchen Elementen das Original zusammengesetzt ist. Hier sind sowohl hierarchische Teilmengenbeziehungen (z. B. Motor und Getriebe bilden einen Antrieb) als auch horizontale Relationsbeziehungen (Ein Getriebe überträgt das Drehmoment eines Motors) zu nennen.
- Die **Eigenschaften** beziehen sich auf Merkmale des Originals. Die Eigenschaften eines Motors sind z. B. seine Nenndrehzahl und der Wirkungsgrad.
- **Semantische und logische Beziehungen** beschreiben Relationen zwischen Entitäten des Originals (z. B. ein Elektromotor ist eine Spezialform eines Motors).
- Das **Verhalten** beschreibt, wie sich Einwirkungen auf das System auswirken (Differentialgleichungen beschreiben z. B. sein dynamisches Verhalten).

Für die sehr heterogenen und umfangreichen Aufgabenstellungen im Entwurfsprozess mechatronischer Systeme kommen verschiedenste Modelle zum Einsatz, die sich sowohl im Zweck als auch im Abstraktionsgrad sehr voneinander unterscheiden können. Modelle sind heutzutage in der Regel immateriell, werden am Rechner erstellt und verfolgen je nach Fachdisziplin unterschiedliche Ziele. Beispielsweise fokussieren die Modelle des Konstrukteurs (z. B. Finite-Elemente-Modelle) die physikalischen Anteile des Systems, primär des Grundsystems und abstrahieren weitestgehend von der Informationsverarbeitung. Modelle des Softwaretechnikers (z. B. Zustandsmaschinen) werden hingegen möglichst unabhängig vom konkreten physikalischen Verhalten definiert. Systemingenieure haben mithilfe von Systemmodellen die komplexen Systemzusammenhänge im Blick. Beim Entwurf mechatronischer Systeme können u. a. folgende (immaterielle) Modelltypen unterschieden werden, bei denen die Modellaspekte (s. o.) verschieden stark ausgeprägt sind.

- Ein **Vorgehensmodell** wie das V-Modell (Bild 2-3) beschreibt die Abfolge von Prozessschritten. Es dient als Hilfsmittel, um diese Prozesse zu planen und zu kontrollieren [Lin09].
- Fachdisziplinübergreifende **Systemmodelle** (z. T. auch als Produktmodelle bezeichnet [VWB+09]) beschreiben die komplexen Systemzusammenhänge und Abhängigkeiten auf einem gleichbleibenden Abstraktionsniveau und z. T.

über alle Phasen des virtuellen Entwicklungsprozesses hinweg [FMS08]. Das Systemmodell gliedert sich dabei zumeist in verschiedene konsistente und verknüpfte Teilmodelle („Partialmodelle“), in denen die jeweiligen Aspekte separat beschrieben werden. Die Systemstruktur wird zum Beispiel in einem Strukturmodell dargestellt.

- **Anforderungsmodelle** bilden die funktionalen und nicht-funktionalen Anforderungen sowie qualitative und quantitative Randbedingungen für das zu entwickelnde Produkt ab [VWB+09]. Dies geschieht meist textuell und informell oder semi-formal.
- Das **Funktionsmodell** legt die funktionale Struktur bzw. Funktionshierarchie des zu entwickelnden Systems fest [VWB+09].
- Ein **Strukturmodell** bildet die hierarchische und modulare Struktur der Komponenten und Teilsysteme des Systems ab.
- **Verhaltensmodelle** beschreiben ein System oder ein Teilsystem z. B. bzgl. seines dynamischen, zeitlichen oder statischen Verhaltens. Quantitative mathematische Modelle ermöglichen die Analyse des Systems durch Simulation.
- In **Gestaltmodellen** wird die Geometrie des Systems beschrieben, zumeist mittels 2D- oder 3D-CAD-Zeichnungen [VWB+09].

Weiterhin unterscheidet man zwischen Modellen, die Systemgrößen und Wechselwirkungen qualitativ darstellen, und solchen, die das System darüber hinaus quantitativ mithilfe eindeutiger mathematischer Größen und Beziehungen wiedergeben [Koh96]. Quantitative Modelle sind Voraussetzung für die numerische Simulation. Hierbei wird weiterhin unterschieden zwischen rein ereignisdiskreten Zustandsänderungen, welche sprunghaft zu diskreten Zeitpunkten auftreten und dazwischen konstant bleiben, und zeitkontinuierlichen Zustandsänderungen, die in einem bestimmten Zeitintervall in unendlicher Anzahl erfolgen können. In hybriden Modellen treten sowohl zeitkontinuierliche als auch zeitdiskrete Zustandsänderungen auf [BR00].

In dieser Arbeit stehen, neben Modellen der semantischen Systemzusammenhänge, vor allem Modelle im Fokus, die das dynamische Verhalten eines mechatronischen Systems beschreiben. Diese beinhalten die in der Regel zeitkontinuierliche Dynamik der Regelstrecke, welche z. B. mechanische, elektrische und/oder hydraulische Bestandteile hat, und den entsprechenden Teil der Informationsverarbeitung zur Beeinflussung der Dynamik (z. B. zur Stabilisierung der Regelungsstrecke). Man spricht in diesem Zusammenhang daher auch von einem sogenannten Multi-Domänenmodell der Dynamik.

2.1.3 Metamodelle

Während Modelle ein Abbild eines Originals darstellen, beschreiben Metamodelle wiederum andere Modelle auf der sogenannten Meta-Ebene. Analog zum Verkürzungsmerkmal in der Definition von STACHOWIAK ist hier das Metaisierungsprinzip entscheidend. Auch der Begriff des Metamodells wird in unterschiedlichen Kontexten verwendet, weshalb es hier ebenso leicht zu Verwirrungen kommen kann, wenn das Metaisierungsprinzip nicht mit genannt wird. STRAHRINGER unterscheidet u. a. zwischen sprachbasierter und prozessbasierter Metaisierung [Str96, Str98].

- **Sprachbasierte Metaisierung:** Das sprachbasierte Metamodell beschreibt die Sprache, in der das Modell des Originals erstellt wird.
- **Prozessbasierte Metaisierung:** Das prozessbasierte Metamodell stellt den Prozess dar, nach dem das Modell erstellt wird. Es handelt sich also um ein Vorgehensmodell.

Die Metaisierung kann in mehreren Ebenen erfolgen, sodass ein Metamodell beschrieben wird durch ein Meta-Metamodell und dieses wiederum durch ein Meta-Meta-Metamodell. Die Object Management Group (OMG) definiert (maximal) 4 Ebenen [OMG07]. Bild 2-2 übertragt diese auf das Beispiel einer einfachen Spiralfeder. In der M0-Ebene liegt diese im Original vor. Das Modell auf der M1-Ebene abstrahiert gemäß Abschnitt 2.1.1 von „uninteressanten“ Eigenschaften, stellt Modellannahmen an und nimmt Vereinfachungen vor. Im dargestellten Fall liegt das Modell der Feder in der Modellierungssprache Modelica vor, deren Sprachkonstrukte auf der nächsthöheren M2-Ebene mithilfe eines Metamodells beschrieben werden. Bei dieser sprachbasierten Metaisierung wird durch das Metamodell beschrieben, dass es die Modelica-Sprachkonstrukte *Model*, *Parameter*, *Equation* etc. gibt. Dasselbe Prinzip wird angewandt, wenn man eine weitere Ebene höher zur M3-Ebene gelangt. Modelle dieser Ebene sind Meta-Metamodelle bezüglich des M1-Modells, jedoch ebenso auch Metamodelle des Modells auf Ebene 2. Folglich ist es spätestens bei einer mehrstufigen Metaisierung wichtig, den Bezug zu klären. Weitere Metaisierungsebenen sind zwar theoretisch denkbar, praktisch aber in der Regel nicht relevant.

2.2 Modellbasierter Entwurf mechatronischer Systeme

Sowohl im disziplinspezifischen Entwurf als auch im übergreifenden Entwurfsprozess mechatronischer Systeme kommen an vielen Stellen unterschiedliche, spezialisierte Modelle zum Einsatz. Die Komplexität und Vielfalt des Entwurfsprozesses erfordert ein systematisches Vorgehen. Neben einigen anderen, zumeist disziplinspezifischen Leitfäden bietet die *VDI-Richtlinie 2206* eine übergreifende methodische Richtschnur [VDI04].

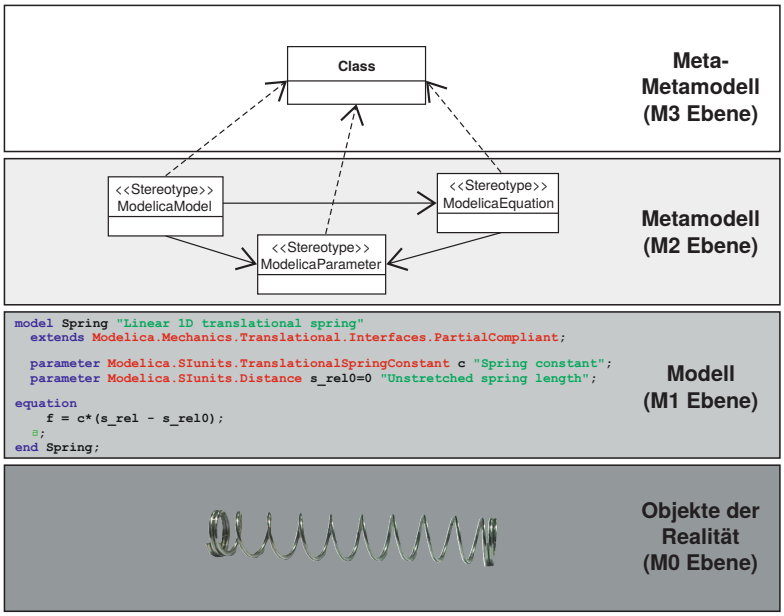


Bild 2-2: Metaisierungsebenen am Beispiel einer Spiralfeder vgl. [OMG07, S. 19]

2.2.1 Die VDI-Richtlinie 2206

Kernelement der VDI-Richtlinie 2206 ist das aus der Softwareentwicklung entnommene *V-Modell*, welches als Vorgehensmodell den Makrozyklus beim Entwurf mechatronischer Systeme darstellt und breite Zustimmung in der Industrie genießt. Es ist als Hilfsmittel zur Orientierung des Entwicklers zu verstehen (vgl. auch [PC85]). Bild 2-3 zeigt eine im Vergleich zur Richtlinie leicht abgewandelte, erweiterte Darstellung. Zu Beginn des Prozesses steht eine Produktidee bzw. ein Entwicklungsauftrag, der das Ergebnis einer strategischen Produktplanung ist. Damit einher gehen auch bereits essentielle Anforderungen an das zu entwickelnde Produkt. In mehreren Durchläufen werden zunächst Labor- und Funktionsmuster, dann Prototypen und schließlich das Produkt entworfen (siehe Bild 2-4).

Das V-Modell gliedert sich in die drei übergeordneten Entwurfsphasen `Systemkonzipierung` (bzw. Systementwurf), `Disziplinspezifischer Entwurf` und `Systemintegration`, welche logisch aufeinander folgen. Sie werden begleitet durch Modellbildung und Analyse der in Abschnitt 2.1 beschriebenen, sehr vielfältigen Modelle. Auf abstrakter Ebene lassen sich in allen drei Phasen die wiederkehrenden Schritte `Zielbeschreibung`, `Synthese` und `Analyse` identifizieren [VDI04]. Sie ha-

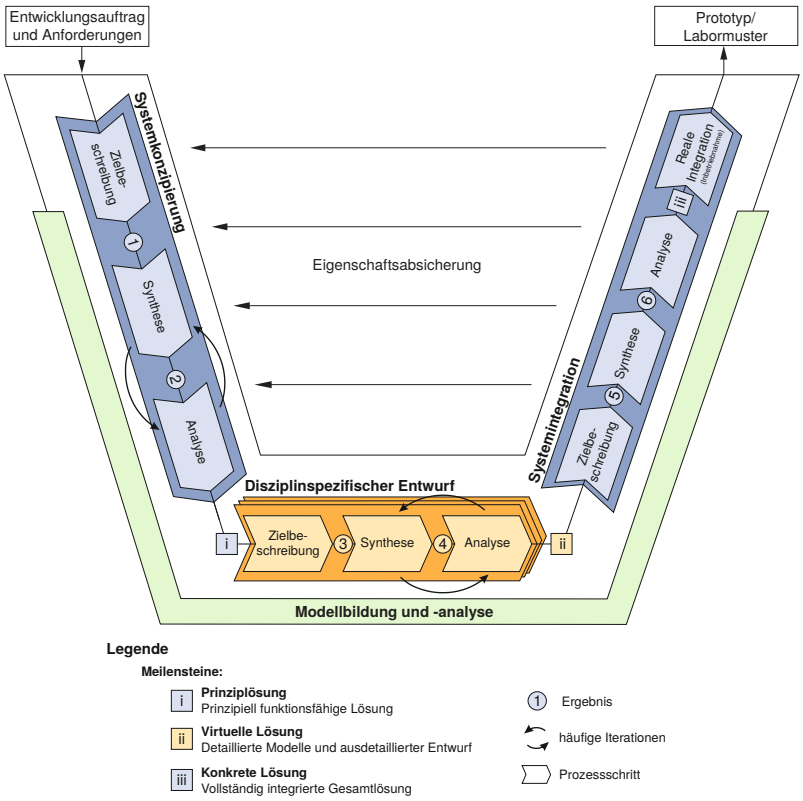


Bild 2-3: Erweiterte Darstellung des V-Modells als Makrozyklus (nach [VDI04], vgl. [Kru, Loc])

ben ihren Ursprung im psychologischen Problemlösungsverhalten [COV86, Dör89] und finden sich in ähnlicher Art und Weise auch im Systems Engineering [HdF+12, S. 76–80].

In der Zielbeschreibung werden auf Grundlage einer Situationsanalyse die Ziele klargestellt und dokumentiert. Anschließend wird während der Synthese ein Lösungsvorschlag erarbeitet. Der Begriff Synthese impliziert hier bereits die Wiederverwendung von Bestehendem. Er bezeichnet im wörtlichen Sinne die Zusammensetzung/Verknüpfung von zwei oder mehreren Elementen zu einer neuen Einheit [KS01, S. 901]. Die Lösung wird dann einer eingehenden Analyse unterzogen und, falls nötig, angepasst oder wieder verworfen. Insbesondere die Synthese- und Analyse-Schritte erfolgen solange iterativ im Wechsel, bis eine zufriedenstellen-

de Lösung erreicht ist. Werden während der Analyse bisher nicht berücksichtigte, aber relevante Aspekte erkannt, so kann jedoch auch die Anpassung oder Vervollständigung der Zielbeschreibung erforderlich sein. Des Weiteren zeigt sich der prinzipiell iterative Charakter des mechatronischen Entwurfs zum einen dadurch, dass das V-Modell zur Erhöhung der Produktreife vom `Labormuster` bis zum `(Vor-)Serienprodukt` immer wieder von Neuem durchlaufen wird (vgl. Bild 2-4). Zum anderen kann es auch in den drei übergeordneten Phasen (Systemkonzipierung, disziplinspezifischer Entwurf, Systemintegration) notwendig sein, in eine frühere zurückzuspringen. In der Regel wird jedoch versucht, die einzelnen Phasen durch Meilensteine von einander abzugrenzen. So ist beispielsweise das Ergebnis der Systemkonzipierung – die Prinziplösung – mit Meilenstein i verbunden. Die Prinziplösung stellt ein Lösungskonzept dar, dessen prinzipielle Funktionsfähigkeit im Zuge der Analyse, z. B. mithilfe von Modellen und Simulationen, nachgewiesen wurde. Auf Basis der Beschreibung der Prinziplösung kann anschließend der disziplinspezifische Entwurf erfolgen. Die beteiligten Disziplinen arbeiten hier soweit möglich parallel und mithilfe spezialisierter Methodiken und Werkzeuge. Das Resultat sind virtuelle (Teil-)Lösungen, die im Zuge der `Systemintegration` zusammengefügt werden. In dieser Phase wird das integrierte System bzgl. der Anforderungserfüllung untersucht. Dies geschieht heute in der Regel schrittweise und mithilfe von Simulationsmodellen. Das Verhalten kann so bereits lange vor der Fertigstellung des ersten realen Prototypen überprüft werden. In-the-loop-Techniken wie Rapid Control Prototyping (RCP)[3], Software-in-the-Loop (SiL)[4] und Hardware-in-the-Loop (HiL)[5] erlauben es, nach und nach immer mehr Komponenten real aufzubauen und zu testen, ehe der erste reale Prototyp in Betrieb genommen wird. Hierfür wird sogenannte Echtzeithardware verwendet, welche die Simulationsergebnisse zuverlässig innerhalb einer festen zeitlichen Schrittweite liefern kann (Echtzeit) [VDI04].

2.2.2 Mechatronische Komposition

Intelligente technische Systeme zeichnen sich in sehr vielen Fällen durch ein kontrolliertes dynamisches Verhalten aus. Das Vorgehen beim modellbasierten Entwurf der Systemdynamik beschreibt die sogenannte *Mechatronische Komposition* [Toe02, Har10]. Sie kann daher als Konkretisierung des disziplinübergreifenden V-Modells bzgl. des Dynamikentwurfs angesehen werden. Während klassischerweise das Grundsystem für den Steuerungs- und Regelungstechniker gegebenen ist,

[3] Die Abkürzung RCP bezeichnet die Kombination eines Modells der Informationsverarbeitung mit einem (teilweise) realen Grundsystem.

[4] Bei SiL-Simulationen wird der fertige (Serien-) Steuerungscode mit einem Simulationsmodell verknüpft.

[5] HiL heißen Simulationen, bei denen die Steuerungshardware oder ein Teil des Grundsystems real vorhanden ist und mit einem Modell kommuniziert.

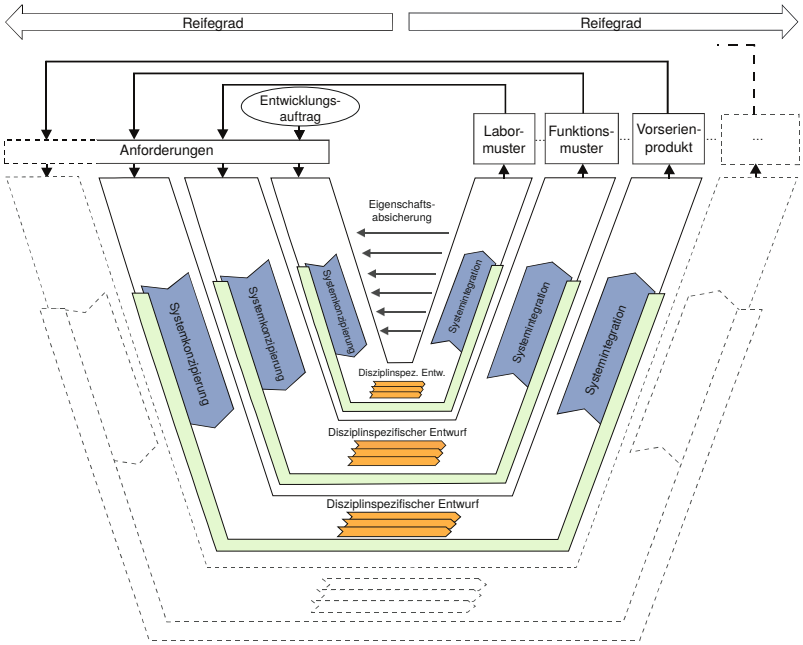


Bild 2-4: Durchlaufen mehrerer Makrozyklen des V-Modells mit zunehmender Produktreife (nach [VDI04])

geht es bei der mechatronischen Komposition um eine ganzheitliche Betrachtung der Systemdynamik, bei der alle relevanten Bestandteile und deren Zusammenspiel gemeinsam, systematisch entworfen werden. Grundlage sind mathematische Modelle zur Beschreibung.

Nach Illg gliedert sie sich die Mechatronische Komposition in die aufeinanderfolgenden Phasen *Komposition des Grundsystems*, *Idealisierte Komposition* und *Ganzheitliche Komposition* [Ill14]. Bei der Komposition des Grundsystems wird sichergestellt, dass das mechanische System die geforderten kinematischen und dynamischen Funktionen erfüllen kann. Stellschrauben zur Auslegung des Systems sind zum Beispiel Parameter wie die Massen, Übersetzungsverhältnisse oder Elastizitäten. Weiterhin wird die Basis für die spätere Regelung/Steuerung des dynamischen Verhaltens gelegt, indem die zu messenden Größen sowie Stellgrößen bestimmt werden [Jus14a]. Im Zuge der nachfolgenden, idealisierten Komposition erfolgt der Entwurf geeigneter Regelstrategien. Zu diesem Zweck wird das Modell um idealisierte Beschreibungen der Aktor- und Sensorprinzipien erweitert. Nun ist es möglich, die Funktionsfähigkeit des mechatronischen Systems bereits zu einem

frühen Zeitpunkt des Entwurfs zu testen, bevor konkrete Lösungselemente – wie zum Beispiel Aktoren und Sensoren – ausgewählt wurden (vgl. Prinziplösung). Für die Suche und Auswahl dieser Elemente können Anforderungen an die dynamischen Eigenschaften abgeleitet werden. Zuletzt wird der Modelldetaillierungsgrad für die ganzheitliche Komposition weiter erhöht, um das integrierte dynamische Verhalten, inkl. zum Beispiel der kompletten Informationsverarbeitung, bzgl. des Führungs- und Störverhaltens zu untersuchen und zu optimieren [Ill14]. Dazu müssen Besonderheiten des Verhaltens (z. B. Nichtlinearitäten) der gewählten Komponenten berücksichtigt werden [Jus14a]. Der Mikroprozess der einzelnen Phasen ist ähnlich wie in der VDI-Richtlinie 2206 als iteratives Wechselspiel zwischen Synthese (Modellbildung, Reglersynthese) und Analyse beschrieben [Toe02, VDI04].

2.2.3 Aufbau von Dynamikmodellen

Im Folgenden soll der Prozess der Modellbildung von Modellen der Systemdynamik, wie er z. B. im Zuge der mechatronischen Komposition zum Einsatz kommt, beschrieben werden (vgl. Bild 2-5). Er setzt an, nachdem die Modellierungsziele, also der Zweck des Modells, geklärt sind.

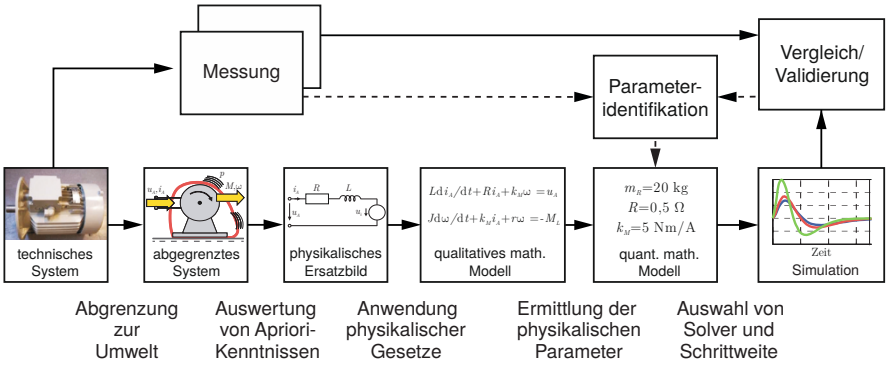


Bild 2-5: Vorgehensweise zur Erstellung von Dynamikmodellen realer Systeme nach [GTS14, S. 40] (vom Autor verändert, vgl. [Loc, Kru])

Im ersten Schritt wird das System von seiner Umwelt abgegrenzt. Dazu werden Schnittstellen, Ein- und Ausgangsgrößen sowie interne Größen definiert. Anschließend erfolgt die Abstraktion, dazu werden Apriori-Kenntnisse über physikalische Zusammenhänge und zulässige Vereinfachungsmöglichkeiten genutzt und ein sog. physikalisches Ersatzbild für das System erstellt. Hierin sind die betrachteten physikalischen Effekte dargestellt. Wie bereits im Abschnitt 2.1.1 erläutert, kommt

diesem Schritt besondere Bedeutung zu, da die Identifizierung der relevanten physikalischen Effekte ausreichendes Systemverständnis und detaillierte Erfahrungswerte voraussetzt. Dies gilt unabhängig davon, wie detailliert das Modell sein soll. Anschließend werden physikalische Gesetze angewendet, um zu einer qualitativen mathematischen Beschreibung zu kommen (mathematisches Modell). Hierbei handelt es sich in der Regel um gewöhnliche Differentialgleichungen (Ordinary Differential Equation (ODE)) oder Differential-Algebraische Gleichungen (Differential Algebraic Equation (DAE)). Indem die Parameter dieser Gleichungen mithilfe von Datenblättern o. Ä. bestimmt werden, gelangt man zum quantitativen mathematischen Modell. Dieses kann dann simuliert, d. h. durch numerische Integrationsverfahren (engl. Solver) gelöst werden. Im Zuge der sog. Modellvalidierung wird abschließend ein Vergleich mit Messungen am realen (Teil-)System angestellt, um die Modellgültigkeit nachzuweisen. Ist der Vergleich nicht zufriedenstellend, kann entweder eine systematische Parameteridentifikation erfolgen oder, falls dies ebenfalls keinen Erfolg bringt, der Detaillierungsgrad erhöht werden (vgl. [Kru]). Existieren für das System noch keine Messungen, weil es sich z. B. noch in der Entwurfsphase befindet, so können diese durch andere Referenzgrößen ersetzt werden, die beispielsweise mithilfe von detaillierten Modellen oder anhand von Funktionsmustern gewonnen werden. Zur Parameteridentifikation und zur Validierung sollten grundsätzlich jedoch verschiedene Referenzgrößen zum Einsatz kommen.

Es existiert eine Vielzahl von Werkzeugen und Sprachen, die den Modellierer bei den beschriebenen Aufgaben unterstützen. Insbesondere bzgl. der Darstellung der mathematischen Modelle bzw. des physikalischen Ersatzbildes lassen sich signifikante Unterschiede feststellen und folgende Typen unterscheiden[6] [GTS14]:

- signalflussorientierte Modellierung (Blockschaltbilder),
- topologieorientierte Modellierung,
- CAD-basierte Modellierung.

Mit Blick auf die Wiederverwendung von Modellen als Lösungswissen eignen sich besonders topologie- und objektorientierte Modellierungsmethoden/-sprachen (z. B. Modelica). Hier werden sog. Ports (Mehrpole) – auch akausale Schnittstellen genannt – mit jeweils einer Fluss- und einer Potentialgröße (Leistungsgrößen) genutzt, was die Verkopplung von abgeschlossenen Teilmodellen unterschiedlicher Disziplinen vereinfacht [MMP+04, LP03]. Zudem erfolgt bereits zum Zeitpunkt der Erstellung des physikalischen Ersatzbildes eine Unterstützung durch das Modellierungswerkzeug. Die mathematischen Gleichungen des Gesamtsystems werden anschließend durch das Simulationswerkzeug (z. B. Dymola) aus den einzelnen

[6] Auf eine genauere Erläuterung der Modellklassen wird an dieser Stelle verzichtet, stattdessen sei z. B. auf [GTS14, S. 40–44] verwiesen.

Komponenten zusammengebaut und optimiert. Hierzu werden generalisierte Kirchhoffsche Gesetze verwendet, bei denen disziplinübergreifende Analogien ausgenutzt werden [GTS14].

2.3 Model-based Systems Engineering

Nach den Anfängen des Systems Engineering bei den Bell Telephone Laboratories in den 40er Jahren [HFK+11] stellte man zum Ende der 1950er Jahre vor allem im Bereich der Luft- und Raumfahrttechnik fest, dass die technischen Systeme derart komplex wurden, dass ihre fehlerfreie Entwicklung auf konventionelle Art und Weise, d. h. mit streng getrennten Organisationseinheiten und Fachdisziplinen, kaum mehr zu leisten war. In dieser Branche kam damals erschwerend hinzu, dass unter extremem Zeitdruck Erfolge verbucht werden mussten. Als Konsequenz wurden Methoden und Werkzeuge entwickelt, die interdisziplinäres, ganzheitliches Systemdenken in Form der Querschnittsdisziplin *Systems Engineering* in den Entwurfsprozess einfließen lassen. Systems Engineering (SE) versteht sich dementsprechend als eine Art „Disziplin der Interdisziplinarität“ oder auch „Metadisziplin“ [Wei07]. Der Dachverband der Systemingenieure, der International Council on Systems Engineering (INCOSE), definiert diese folgendermaßen:

> *Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem:*
>
> - *operations,*
> - *cost and schedule,*
> - *performance,*
> - *training and support,*
> - *test,*
> - *manufacturing and*
> - *disposal.*
>
> *Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs. [HFK+11, S. 7]*

Hier wird deutlich, dass es nicht nur um eine Systemgestaltung, sondern vielmehr um die integrative System- und Projektbetrachtung geht [GDS+13]. Aufgaben des Systems Engineerings sind laut [Wei07, S. 9] u. a.:

- Anforderungsdefinition, -analyse und -management,
- Systementwurf,
- Systemintegration,
- Systemverifikation und -validierung,
- Projekt- und Risikomanagement.

Sie beziehen sich ausdrücklich „nur“ auf die Systemebene und haben nicht den Anspruch, in die detaillierten disziplinspezifischen Entwicklungsprozesse einzugreifen, wenngleich sich in letzteren durchaus Denkweisen des Systems Engineering wiederfinden lassen [Wei07, S. 8]. Mit dem Zusatz „Model-based“ wird verdeutlicht, dass ein Systemmodell (vgl. Abschnitt 2.1.2) für die Aufgaben verwendet und in den Mittelpunkt gestellt wird. Zwar gibt es mit SysML eine etablierte Modellierungssprache für Systemmodelle im Systems Engineering, jedoch gilt dies nicht für eine anzuwendende Methodik [GDS+13]. Im Folgenden werden zunächst einige der existierenden methodischen Konzepte sowie anschließend die im Kontext dieser Arbeit wichtigen Aspekte der Sprachen SysML/UML und CONSENS vorgestellt.

2.3.1 Konzepte und Vorgehensweisen

Im Bereich des Systementwurfs setzen vor allem Hersteller von Softwarewerkzeugen wie Dassault Systèmes und Siemens auf den sogenannten *RFLP-Ansatz*. Hinter diesem Kürzel verbergen sich vier Schritte, die während des Produktlebenszyklus abgearbeitet werden:

- **Requirements:** Erfassung, Management und Verfolgung von Anforderungen,
- **Functional Design:** Abbildung der funktionalen Zusammenhänge in Funktionsstrukturen,
- **Logical Design:** Beschreibung logischer Zusammenhänge und u. a. auch Modellierung zur Simulation des dynamischen Verhaltens (z. B. auf Basis von Modelica),
- **Physical Design:** CAD-Modellierung z. B. der Konstruktion.

Diese Schritte werden systematisch mithilfe entsprechender Werkzeugunterstützung abgearbeitet. Teilweise wird dabei auf eine gemeinsame Datenbank zurückgegriffen [KK13].

Die Spezifikationstechnik *CONSENS (CONceptual design Specification technique for the ENgineering of complex Systems)* basiert auf den etablierten Methoden der Konstruktionslehre [PBF+13] sowie der VDI-Richtlinie 2206 und definiert einerseits ein Vorgehen zur Erstellung eines fachdisziplinübergreifenden Systemmodells

in der Konzipierungsphase (siehe Bild 2-6), andererseits aber auch Sprachelemente für das Systemmodell. Dies besteht je nach Ausprägung aus bis zu zehn Aspekten bzw. Partialmodellen [GTS14, GFD+08]. Das enthaltene Vorgehensmodell schlägt vor, mit einer `Umfeldbeschreibung` sowie der Beschreibung von `Anwendungsszenarien` und `Anforderungen` zu beginnen. Auf der Basis werden die `Funktionshierarchie`, die Systemstruktur (`Wirkstruktur`) und die `Gestalt` in Form von z. B. einer Prinzipskizze erarbeitet. Die `Wirkstruktur` bildet den Kern der Prinziplösung. Vervollständigt wird das Systemmodell durch die Aspekte `Zustände`, `Aktivitäten` und `Sequenzen`, welche das Verhalten des mechatronischen Systems beschreiben [IKD+13, GKP+10].

Bild 2-6: Die Aspekte von CONSENS in ihrer groben Reihenfolge [IKD+13]

Das Vorgehen *SYSMOD* nach Weilkiens stellt einen Vorschlag für einen Bau-/Werkzeugkasten zur Nutzung der SysML dar. Es beschreibt die Aktivitäten und Ergebnisse von der Produktidee bis zum Systementwurf und wie hierfür jeweils die Sprachelemente der SysML verwendet werden können. Mithilfe von Aktivitätsdiagrammen werden zwei übergeordnete Vorgehensschritte definiert [Wei07].

1) **Analyse:** Hier wird zunächst der Projektkontext beschrieben und Anforderungen ermittelt. Auf dieser Basis werden der Systemkontext und Anwendungsfälle modelliert sowie ein Glossar erstellt. Zuletzt werden die strukturellen Zusammenhänge zwischen den identifizierten fachlichen Objekten des Systems dargestellt.
2) **Design:** Die identifizierten Anwendungsfälle sollen anschließend in die Realität umgesetzt werden. Dazu werden zuerst die Interaktionen zwischen dem System und anderen Akteuren (Personen oder andere Systeme) beschrieben, um dann die Systemschnittstellen zur Außenwelt ableiten zu können. Auf dieser Grundlage erfolgt die Modellierung der Systemstruktur und des Systemverhaltens.

Daneben finden sich noch die Methoden FAS (functional architecture for systems) nach Lamm et al. [LW10] und OOSEM (object oriented systems engineering method) [FMS12]. Weiterhin geben das *System Engineering Handbook* der INCOSE [HFK+11] sowie das *Cookbook for MBSE with SysML* [KWH+11] eine ganze

Reihe von methodischen Hinweisen zum Systems Engineering. Einen abstrakteren Überblick über die SE-Philosophie und Vorgehensmodelle (prozessbasierte Metamodelle) sowie Methoden zum Projektmanagement geben Haberfellner et al. [HdF+12].

2.3.2 Disziplinübergreifende Modellierungssprachen

Bei beiden hier vorgestellten disziplinübergreifenden Modellierungssprachen handelt es sich um grafische Sprachen. Das bedeutet, dass sie Diagramme als Ausdrucksmittel nutzen. Die Bedeutung der grafischen Symbole und Elemente ist klar definiert. Aufgrund der großen Komplexität und der unterschiedlichen Aspekte der Systeme ist es in der Regel nicht möglich und mit Blick auf die Interpretierbarkeit auch nicht zielführend, ein einzelnes, großes Diagramm zu verwenden. Aus diesem Grund werden für die unterschiedlichen Zwecke verschiedene Diagramme – auch Sichten genannt – verwendet. Diese beziehen ihre Elemente aus einem gemeinsamen Modell, welches die volle Beschreibung des Systems beinhaltet. Das kann zum Beispiel eine Datenbank sein, in der alle Elemente und deren Zusammenhänge gespeichert sind. Die Sichten stellen Projektionen des Modells dar, die es dem Anwender von einer bestimmten Perspektive aus zeigen und Aspekte weglassen, die in diesem Zusammenhang unwichtig sind (vgl. Bild 2-7) [Alt12, Wei07].

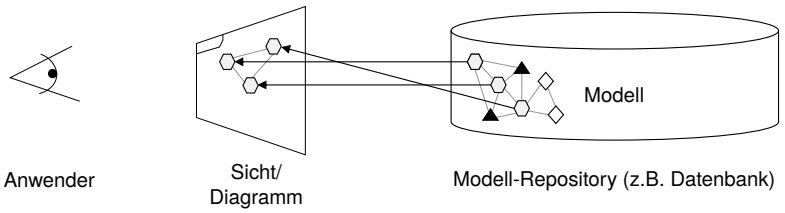


Bild 2-7: Unterschied zwischen Modell und Sicht nach [Alt12, S. 39] (vom Autor verändert)

SysML/UML

Die *Unified Modeling Language (UML)* definiert ebensolche Sichten auf ein Modell mithilfe verschiedener Diagrammarten [ISO05]. Sie wird durch die Object Management Group (OMG) spezifiziert. Im klaren Fokus der UML stehen Softwaresysteme. In der Disziplin Softwareentwicklung ist sie, aufgrund ihrer sehr allgemein gehaltenen Art, universell einsetzbar und daher zur dominierenden Modellierungssprache geworden. Es stehen sehr umfangreiche Ausdrucksmittel zur Verfügung, von denen nicht immer alle angewendet werden müssen. Darüber

hinaus lässt sich die UML über den sogenannten Profilierungs-Mechanismus sowie durch sogenannte Stereotypen einschränken und erweitern, um sie so besser an die Aufgabe anzupassen. Ein solches Profil zum Zuschnitt der UML, das es ermöglicht, sie für den Aufbau eines Systemmodells zu nutzen, ist die SysML [Alt12, S. 32]. Die in der in SysML-Spezifikation [OMG12] definierten Diagramme stellen somit eine Kombination aus modifizierten UML-Diagrammen und einigen wenigen SysML-spezifischen Diagrammen (z. B. Parameterdiagramm) dar (vgl. Bild 2-8). Sie können in die Kategorien Struktur-, Verhaltens- und Anforderungsdiagramme einsortiert werden.

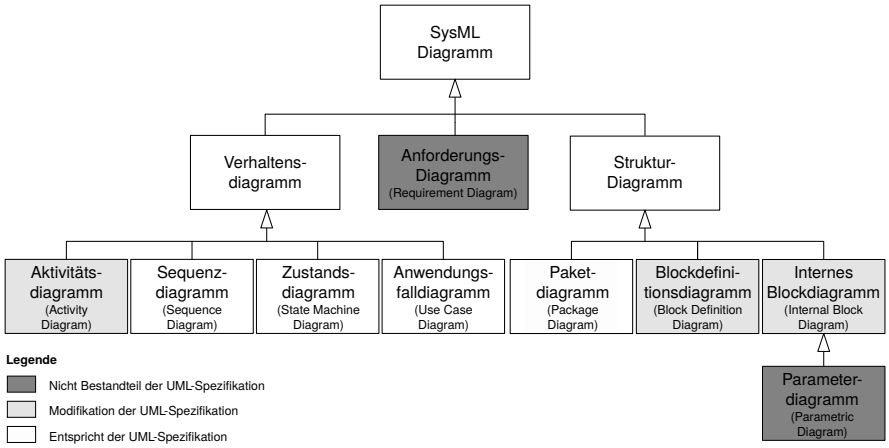


Bild 2-8: Übersicht der SysML Diagramme nach [Wei07, FMS12, OMG12]

Im Folgenden sollten die beiden für diese Arbeit wichtigen Diagramme der SysML bzw. UML, das Blockdefinitionsdiagramm/Klassendiagramm und das Aktivitätsdiagramm, kurz erläutert werden. Für eine umfassendere Darstellung sei auf die entsprechende Literatur verwiesen [Wei06, Wei07, FMS12].

Das **Blockdefinitionsdiagramm (bdd)** der SysML ist eine Modifikation des UML-Klassendiagramms. Um die starke assoziative Verknüpfung mit der Softwareentwicklung abzuschwächen, wird es in der SysML-Spezifikation umbenannt und dient nun analog dazu, allgemeingültige Aussagen über Systembausteine und deren statische Struktur zu modellieren. Die Diagrammelemente und ihre Semantik unterscheiden sich jedoch nur minimal. Systembausteine werden im Blockdefinitionsdiagramm als Rechtecke mit einzelnen, optionalen Abteilungen für Bausteinattribute, also konstante oder z. B. zeitlich veränderliche Werte (values), Kompositionen (parts), Zusicherungen (constraints) und/oder Referenzen (references), dargestellt. Mithilfe von Assoziationen sowie Kompositions- und Ag-

gregationsbeziehungen werden alle gültigen Strukturbeziehungen zwischen zwei Systembausteinen beschrieben [Wei06]. Da es sich um ein Blockdefinitionsdiagramm handelt, gelten die Angaben immer für alle Instanzen dieses Typs.

Dies soll nun anhand eines Spiralkneters zum industriellen Kneten von Brot- und Brötchenteig verdeutlicht werden (siehe Bild 2-9 links; vgl. auch Abschnitt 3.2 auf Seite 79). In dem dargestellten Diagramm ist modelliert, dass der Systembaustein `Spiralkneter` (also jeder Spiralkneter) einen Wert für die Gesamtmasse `m_ges` vom Typ `Masse` besitzt. Wie groß diese Masse ist, ist hier jedoch noch nicht festgelegt. Weiterhin sind alle Spiralkneter vom Typ `Chargen-Kneter` (Generalisierung) und bestehen immer jeweils aus einem Element der Typen `Bottich`, `Antrieb` und `Knetspirale`. Dies wird durch die Kompositionsbeziehung und die angegebene Multiplizität an deren Ende ausgedrückt. Der obligatorische Antrieb ist der für die Knetspirale, das Fehlen eines Antriebs in der Rolle des Bottichantriebs führt nicht dazu, dass es sich nicht mehr um ein Element vom Typ des Systembausteins `Spiralkneter` handelt (Aggregationsbeziehung). Weiterhin wird durch die Assoziation mit dem entsprechenden Block ausgedrückt, dass bis zu fünf Spiralkneter von einem `Bäcker` bedient werden können.

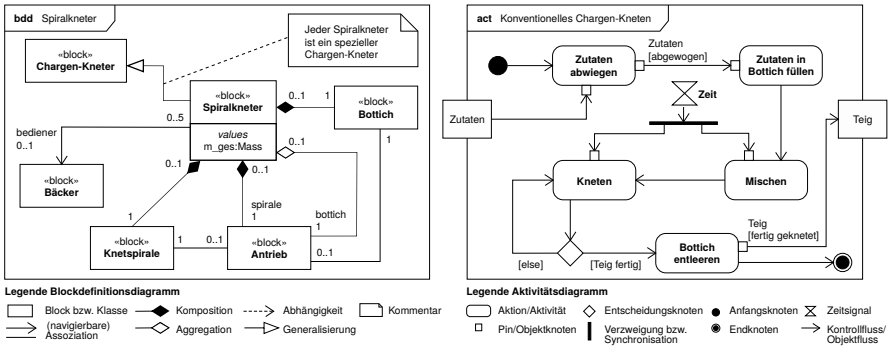


Bild 2-9: Blockdiagramm (links) und Aktivitätsdiagramm (rechts) am Beispiel eines Spiralkneters

Mithilfe eines **Aktivitätsdiagramms (act)** können Abläufe beschrieben werden. Eine Aktivität beschreibt hierbei immer mehrere elementare Aktionen bzw. untergliederte Aktivitäten, die sequentiell oder parallel abgearbeitet werden. Die SysML ermöglicht die erweiterte Nutzung nicht nur für Algorithmen und Operationen, sondern auch zur Modellierung von Anwendungsfällen oder (Geschäfts-) Prozessen, indem zum Beispiel ein kontinuierlicher Fluss (stream) modelliert werden kann. Das Diagramm besteht aus Knoten für Aktionen oder Aktivitäten und Kanten für den Übergang zwischen ihnen. Es wird vom Startknoten bis zum

Endknoten durchlaufen (Tokenfluss). Hierbei wird zwischen Kontrollfluss und Objektfluss differenziert, der Unterschied ist jedoch im Diagramm nur am Kontext festzumachen. Im Gegensatz zu Kontrollflüssen können Objektflüsse Werte, Parameter oder allgemein Objekte übertragen. Sie erfolgen zwischen den Ein- und Ausgabeschnittstellen (Pins) der Aktivitäten [Wei06].

Bild 2-9 (rechts) zeigt das Beispiel des Ablaufs beim `konventionellen Chargen-Kneten` gezeigt, welches einige Elemente des Aktivitätsdiagramms enthält. Nach dem Beginn der Aktivität am Startknoten müssen die Zutaten, welche am Eingabe-Pin der Aktivität anliegen, zunächst abgewogen werden. Ihr Zustand ändert sich folglich zu *abgewogen*, was durch die eckigen Klammern an der Kante verdeutlicht wird. Anschließend werden sie in den Bottich gefüllt, worauf das `Mischen` starten kann. Diese Aktivität hat zusätzlich einen Eingabe-Pin, der von einem Zeitsignal gespeist wird. Sofern die Mischzeit abgelaufen ist, wird zum `Kneten` gewechselt. Wird diese Aktivität verlassen, weil die Knetzeit abgelaufen ist, so muss geprüft werden, ob der Teig fertig ist. Falls die Bedingung `Teig fertig` nicht erfüllt ist, muss weiter geknetet werden. Sofern sie wahr ist, wird zur Aktion `Bottich entleeren` gewechselt und anschließend die Aktivität beendet sowie der fertige Teig als Ausgabeobjekt bereitgestellt.

CONSENS-Partialmodelle

Bild 2-10 zeigt sieben kohärente Sichten/Partialmodelle von CONSENS[7] am Beispiel kooperierender Delta-Roboter [GTS14]. Im `Umfeldmodell` wird die Beziehung des Systems zu seinem Umfeld beschrieben. Dazu wird es als „Black Box“ angesehen und Einflüsse mittels Energie-, Stoff- oder Signalflüssen von und zu umgebenen Systemen/Umfeldelementen modelliert. Mithilfe von textuellen Beschreibungen werden `Anwendungsszenarien` spezifiziert. Sie charakterisieren das gewünschte Systemverhalten in verschiedenen Situationen. Mittels einer semiformalen tabellarischen Auflistung werden anhand dieser `Anforderungen` an das zu entwickelnde Produkt erfasst, welche sowohl quantifizierbar als auch abstrakter Natur sein können. Auf Grundlage der ersten drei Partialmodelle erfolgt die Modellierung der `Funktionshierarchie`. Hierbei wird die Hauptfunktion solange in Teilfunktionen unterteilt, bis sich ein geeignetes Lösungsprinzip zu ihrer Erfüllung findet. Mithilfe der gefunden Lösungsprinzipien – auch Lösungsmuster genannt – kann dann die `Wirkstruktur` erstellt werden. Darin werden die Beziehungen zwischen Systemelementen sowie deren Merkmale beschrieben. Systemelemente können Module, Bauteile oder Softwarekomponenten sein, die über Stoff-, Energie- und Informationsflüsse in Wechselwirkung stehen oder logisch miteinander verknüpft sind. Im Partialmodell `Verhalten` werden mit Aktivitäten, Zuständen und Sequenzen die Aktionen und Reaktionen des Systems bzw. der Systemsoftware zur

[7] Das oben beschriebene Vorgehen sieht das Durchlaufen im Uhrzeigersinn vor.

Erfüllung der spezifizierten Anwendungsszenarien konzipiert. Der Aspekt `Gestalt` beinhaltet eine grobe Darstellung des Erscheinungsbilds sowie Anzahl, Form, Lage, Anordnung und Art der Wirkflächen und Wirkorte. Die Gestalt kann zunächst mittels Handskizzen, aber auch mithilfe gängiger 3D-CAD-Systeme konzipiert werden [GTS14].

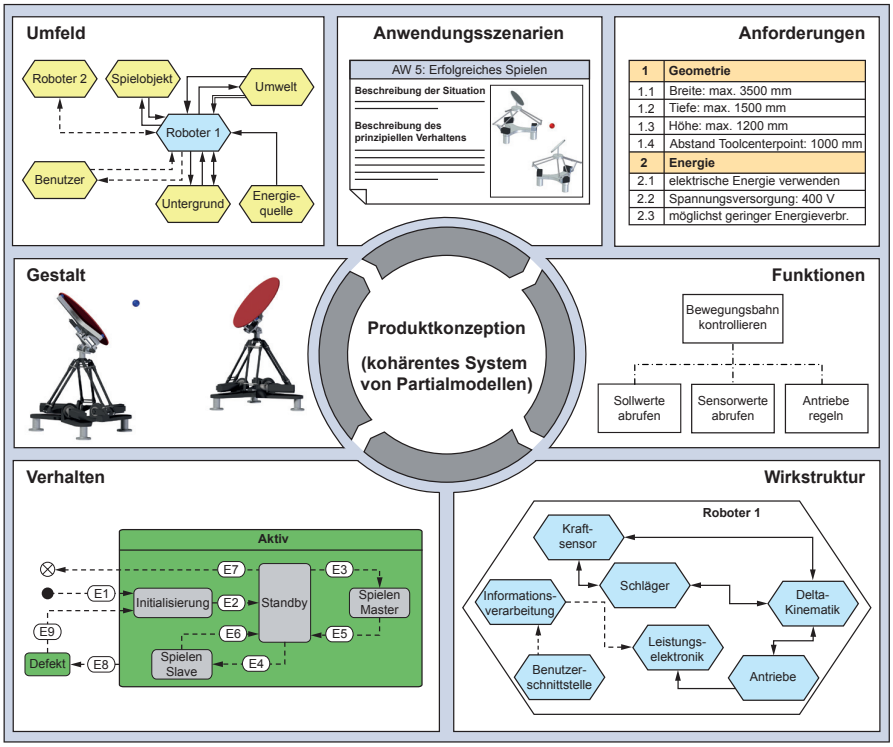


Bild 2-10: Partialmodelle der Spezifikationstechnik CONSENS am Beispiel kooperierender Delta-Roboter [GTS14, S. 38]

2.4 Semantische Technologien und wissensbasierte Systeme

„Wissen ist [...] die Fähigkeit, Daten im Kontext zu interpretieren“ [Den12, S. 4]. Damit Rechner nicht nur Daten, sondern auch Wissen verarbeiten können, muss dieses Wissen in geeigneter Weise aufbereitet bzw. modelliert werden, nämlich

mithilfe formaler Strukturen. Hierzu muss zunächst zwischen der Syntax und der Semantik einer Information unterschieden werden. Während mit der Syntax die Art der Zeichenfolge gemeint ist, welche einem vorbestimmten Zeichensystem folgt, wird die Bedeutung der Zeichenfolge als Semantik bezeichnet. Ein Beispiel:

Massenträgheiten vereinfachen die Suche nach Lösungselementen.

Die Syntax dieses Satzes ist vollkommen korrekt. Alle Wörter sind richtig geschrieben, am Ende steht ein Punkt. Auch die Struktur des Satzes mit Subjekt, Prädikat und Objekt stimmt. Dennoch erschließt sich die Bedeutung – d. h. die Semantik – für den Leser zunächst nicht, zumindest nicht ohne den Zusammenhang. Rechner und Textverarbeitungsprogramme können dies jedoch nicht ohne weiteres beurteilen. Sie können den Satz lediglich hinsichtlich der Einhaltung von korrekter Syntax und einfachen Grammatikregeln überprüfen.

Der Ausdruck *semantische Technologien* ist als Oberbegriff zu verstehen. Ganz allgemein stellen semantische Technologien Mittel bereit, um die Bedeutung einer Information in der Art auszudrücken, dass sie sowohl für einen Menschen als auch für eine Maschine eindeutig verständlich ist. Vor diesem Hintergrund bieten sie großes Potential bei der Entwicklung intelligenter technischer Systeme, da sie zum einen die Kommunikation zwischen den verschiedenen Fachexperten der beteiligten Disziplinen vereinfachen können, die von unterschiedlichen Denkweisen und Begriffswelten geprägt sind. Zum anderen ermöglichen sie eine noch bessere Unterstützung des komplexen Entwurfsprozesses durch Maschinen.

2.4.1 Kollaboratives Wissensmanagement mithilfe wissensbasierter Systeme

Das Auffinden, Bewahren, Nutzen und Pflegen von Wissen ist heute ein wesentlicher Wettbewerbsfaktor, bei dem es jedoch in vielen Unternehmen noch großen Nachholbedarf gibt [SHJ+06, Bai08]. Oftmals ist Wissen nur in den Köpfen oder in individuellen Wissensspeichern (E-Mail-Ordner, Dokumentensammlungen) der Mitarbeiter verfügbar. Zum Teil existieren zentrale Daten- und Dokumentenspeicher, die jedoch nur schwer zu durchsuchen sind.

Wissensbasierte Systeme sind intelligente Informationssysteme, die ein kollaboratives Wissensmanagement auf Basis von formalisiertem Wissen unterstützen und ermöglichen sollen [GTS14, S. 58]. In wissensbasierten Systemen sind die Komponenten zur Abbildung/Modellierung von Wissen (die Wissensbasis) in der Regel streng von jenen Bestandteilen getrennt, die zur Nutzung, Auswertung, Erweiterung und Erklärung des Wissens dienen (vgl. [Den12]). Bild 2-11 zeigt einen verallgemeinerten, schematischen Aufbau. Kern ist die Wissensbasis, die

sowohl Fakten zur *Repräsentation* von Wissen (engl. knowledge representation) als auch sog. *Inferenzmechanismen* enthält. Inferenzmechanismen sind Regeln oder Vorschriften, welche die Schlussfolgerung von neuen Fakten ermöglichen. Des Weiteren hat sich die Unterteilung der Wissensbasis in *Terminological Box (T-Box)* und *Assertional Box (A-Box)* etabliert. Die zugrundeliegende Denkweise entspricht im Grundsatz der sprachbasierten Metaisierung (vgl. Kapitel 2-2). Die terminologischen und eher allgemeingültigen Inhalte in der T-Box dienen als Grundlage („Sprache“) zur Formulierung der konkreten Aussagen und Fakten in der A-Box (vgl. [HKR+08]). Dementsprechend beinhaltet die T-Box vor allem übergeordnete Konzepte und Regeln, während die A-Box überwiegend konkretes Faktenwissen über Individuen enthält. Die Grenzen zwischen diesen beiden Teilen der Wissensbasis verschwimmen zwar oftmals, in der Praxis hat sich jedoch gezeigt, dass die Unterteilung und die damit verbundene Denkweise im Grundsatz dennoch sehr sinnvoll sind, insbesondere beim Aufbau der Wissensbasis.

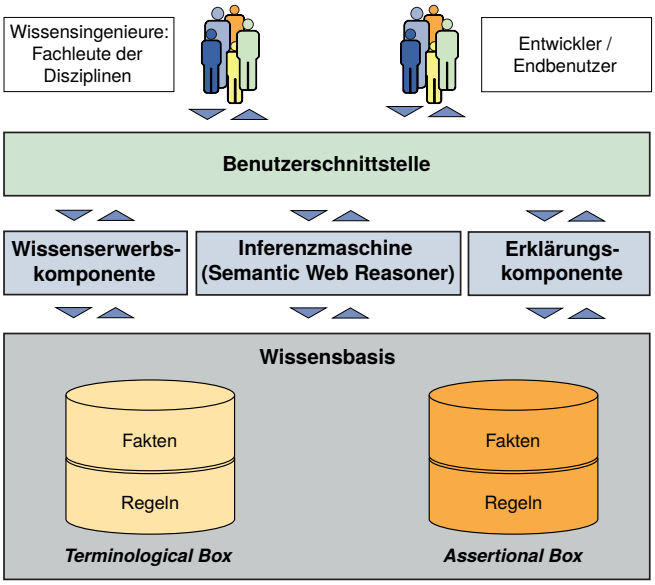


Bild 2-11: Schematische Übersicht eines wissensbasierten Systems vgl. [GTS14, S. 58] (vom Autor verändert)

Getrennt von der Wissensbasis werden die Komponenten zur Wissensverarbeitung aufgebaut. Hierzu zählen:

- die **Inferenzmaschine**, die das permanente Wissen (Fakten und Regeln) der Wissensbasis auswertet und fallspezifische Schlussfolgerungen zieht,

- die **Erklärungskomponente**, welche dem Nutzer die auf Schlussfolgerungen basierenden Antworten des Systems bei Bedarf erläutert,
- die **Wissenserwerbskomponente**, die es Experten und Wissensingenieuren ermöglicht, den Inhalt der Wissensbasis zu erweitern und zu pflegen,
- und die **Benutzerschnittstelle**, welche die Kommunikation mit dem Expertensystem möglich macht.

Ggf. kann bei Letzterer noch zwischen separaten Schnittstellen für Wissenserwerb/ Wartung und Nutzung des Systems unterschieden werden (vgl. auch [BKI08]).

Auch wenn die Wichtigkeit und das Potential von effizientem Wissensmanagement nahezu unbestritten sind, lässt sich gerade bei kleinen und mittleren Unternehmen (KMU) noch erheblicher Nachholbedarf bei der Einführung wissensbasierter Systeme feststellen [Bai08]. Neben technischen Problemen gestaltet sich vor allem der Wissenserwerb schwierig. Es existieren zwar verschiedene Vorgehensweisen zur „Wissenserzeugung“ im Unternehmen, wie zum Beispiel das sog. *SECI-Modell (Socialisation, Externalization, Combination, Internalization)* nach Nonaka und Takeuchi [NT97], das kollaborative Wissensmanagement mithilfe wissensbasierter Systeme wird aber dennoch durch wesentliche Probleme erschwert (vgl. [Wag06]):

- neues Wissen zu erfassen sowie Fehler zu finden und zu korrigieren bedeutet großen Aufwand (sehr verschiedene Quellen und Datenmodelle),
- die Informationskanäle zwischen den Wissensquellen und der Wissensbasis sind schmal (wenige Experten, Dokumente etc.),
- neues Wissen wird oft erst mit Verzögerung in das zentrale System eingepflegt (Latenz der Akquisition),
- bei der Wissenserfassung und -repräsentation entstehen leicht Ungenauigkeiten und Fehler.

Wagner schlägt daher eine Wissensakquisition vor, die – angelehnt an Wikis und Open Source Software – auf Beiträge vieler Nutzer setzt [Wag06]. Andere Ansätze zielen verstärkt auf die Wiederverwendung und Erweiterung bestehender Wissensbasen oder deren automatischen Aufbau ab. Studer et al. sowie Subhashini und Akilandeswari geben eine Übersicht über Prinzipien und Methoden [SBF98, SA11].

2.4.2 Die Idee des Semantic Web

Die ersten Konzepte des Semantic Web entstanden zu Beginn der 2000er Jahre und wurden wesentlich durch Tim Berners-Lee, der auch schon als Begründer des Internets gilt, vorangetrieben [BHL01]. Ziel ist die schrittweise Erweiterung

des World Wide Web (WWW) hin zu einem „einzigen Bedeutungsnetz von allem und jedem“ [BF99], dem Semantic Web. Das Semantic Web stellt also quasi eine globale, dezentrale Wissensbasis dar. Dazu werden neben den konventionell aufbereiteten Inhalten auch durch Maschinen analysierbare Informationen über deren „Bedeutung“ online zur Verfügung gestellt. Computer sollen die Inhalte „verstehen“, interpretieren und in Verbindung setzen können. Das geschieht durch das Annotieren von Metadaten[8]. Um das Wissen nutzen zu können, muss es jedoch nicht nur abgebildet, sondern auch ausgewertet werden. Dies übernehmen spezialisierte Softwareprogramme, auch Agenten genannt. Sogenannte *Crawler* durchforsten (engl. to crawl) das Semantic Web nach Inhalten aus unterschiedlichsten Quellen auf der Suche nach relevanten Informationen, die helfen die jeweilige Fragestellung zu beantworten. *Semantic Web Reasoner* ziehen logische Schlussfolgerungen im Sinne einer Inferenzmaschine. Durch die maschinenverständliche Abbildung der Inhalte können verschiedene Agenten Informationen selbst dann austauschen, wenn sie nicht explizit für eine Zusammenarbeit konzipiert wurden. Schlüsseltechnologie für den Aufbau des Semantic Web sind die sog. Ontologien.

2.4.3 Ontologien

Ontologien stellen eine Möglichkeit dar, Informationen über die semantischen Zusammenhänge zwischen Begriffen zu modellieren und sie durch die formale Beschreibung für den Rechner auswertbar zu machen. Sie werden vor allem in der Semantic Web-Community genutzt und weiterentwickelt. Die zugrundeliegende Annahme ist, dass die Bedeutung eines Begriffs (Objekts) vor allem durch seine assoziativen Beziehungen zu anderen Begriffen (Objekten) definiert wird [Den12].

Der Begriff „Ontologie“ ist entliehen aus der Philosophie, wo er ursprünglich die Lehre des „Seins“ bezeichnet. Immanuel Kant definiert:

> *„Die Ontologie ist diejenige Wissenschaft (als Teil der Metaphysik), welche ein System aller Verstandesbegriffe und Grundsätze, aber nur, sofern sie auf Gegenstände gehen, welche den Sinnen gegeben und also durch Erfahrung belegt werden können, ausmacht. Sie berührt nicht das Übersinnliche, welches doch der Endzweck der Metaphysik ist, gehört also zu dieser nur als Propädeutik, als die Halle oder der Vorhof der eigentlichen Metaphysik und wird Transzendental-Philosophie genannt, weil sie die Bedingungen und ersten Elemente aller unserer Erkenntnis a priori enthält.“*[9]

[8] Analog zu den Meta-Modellen (siehe Kapitel 2.1) handelt es sich hierbei um „Daten über Daten“

[9] Aus Kants "Über die Fortschritte der Metaphysik seit Leibniz und Wolf", herausgegeben 1804

In den 1990er Jahren wurde der Begriff in der Informatik und vor allem im Bereich der künstlichen Intelligenz aufgegriffen [Gru93]. Hier bezeichnet er einen mehrdimensionalen Graphen aus Knoten und Kanten, vergleichbar mit einem semantischen Netz, das genutzt wird, um Wissen formal aufzubereiten. Dazu wird nicht nur die Syntax eines Begriffes, sondern auch seine Bedeutung – die Semantik – abgebildet. Es handelt sich analog zu obiger Definition um ein System aus (Verstandes-)Begriffen, grundsätzlichen, logischen Beziehungen und expliziten Werten. Die Begriffe werden in Form einer Taxonomie klassifiziert und mithilfe assoziativer Relationen in Verbindung gesetzt. Hinzu kommen Inferenzregeln, die logische Schlussfolgerungen ermöglichen. Durch die Mehrdimensionalität ergeben sich signifikante Vorteile gegenüber einer eindimensionalen, relationalen Datenbank. So ist es möglich, neben dem übergreifenden Konsens verschiedene Sichten auf dasselbe Wissen in einem semantischen Netz zu modellieren und logisch miteinander zu verknüpfen. Diese Verknüpfungen können im Anschluss durch sog. Reasoner (Inferenzmaschinen) automatisch ausgewertet und somit implizit vorliegendes Wissen explizit nutz- bzw. abfragbar gemacht werden. Gegenüber anderen semantischen Modellen zeichnen sich Ontologien durch größere Reichhaltigkeit aus, denn Schlüsse können bereits auf Modellebene und nicht nur durch auswertende Agenten gezogen werden [BP06]. Sie sind damit eine wichtige semantische Technologie und eine Grundvoraussetzung für den Aufbau des Semantic Web (vgl. Abschnitt 2.4.2). Wesentlich für Ontologien ist, dass sie gemeinschaftlich festgelegte Begriffe formal festhalten [Den12, RH06]. Dadurch stellen sie einen maschinell auswertbaren Wissenskonsens dar, der sowohl die Kommunikation zwischen Maschinen, Mensch und Maschine, aber auch zwischen Menschen in den modellierten Bereichen unterstützt [Pic03].

Da es sich bei einer Ontologie allgemein zunächst „nur“ um ein formales Wissensmodell handelt, ergeben sich vielfältige Anwendungsmöglichkeiten. Blumauer und Pellegrini nennen in [BP06] beispielhaft folgende Ziele:

- Ermöglichung von Datenaustausch zwischen Programmen,
- Vereinheitlichung und Übersetzung zwischen verschiedenen Wissensrepräsentationsformen,
- Entwicklung von Services zur Unterstützung von Wissensarbeitern,
- Abbildung und Veranschaulichung von Theorien,
- Ausdrücken der Semantik strukturierter und semi-strukturierter Informationen,
- Unterstützung der Kommunikation zwischen Menschen.

Hieraus haben sich verschiedene, z. T. widersprüchliche Ansätze zur Klassifizierung von Ontologien entwickelt (vgl. [RH06, Hen12]). Folgende Charakterisierungen und

Begriffe finden sich allerdings an vielen Stellen und seien daher hier exemplarisch genannt.

- **High-Level Ontologies** beschreiben generalisierte und abstrakte Begriffe/Konzepte (engl. concepts) wie Raum oder Zeit, welche unabhängig von einer Domäne bzw. Disziplin sind. Sie vereinheitlichen diese für große Benutzergruppen [Gua98].
- **Domain Ontologies** stellen wiederverwendbare Ontologien einer bestimmten Domäne bzw. Disziplin (z. B. Maschinenbau, Elektronik etc.) dar. Sie modellieren das Vokabular sowie die Begriffe und Zusammenhänge dieser Disziplin [Gua98, Fen04].
- **Task Ontologies** modellieren in ähnlicher Art und Weise Vokabular und Konzepte zur Lösung einer konkreten Aufgabe [Gua98, Fen04]. Diese müssen nicht zwangsläufig einer einzelnen Domäne zuordenbar sein.
- Mit dem Begriff **Lightweight Ontologies** werden Ontologien auf konzeptueller Ebene bezeichnet. Sie beinhalten Taxonomien, Merkmale und übergeordnete Beziehungen zur Beschreibung von Konzepten [RH06].
- Im Gegensatz dazu wird in **Heavyweight Ontologies** bestimmtes Wissen modelliert, indem Axiome und Bedingungen hinzugefügt werden. Diese konkretisieren die Bedeutung der Terme einer Lightweight Ontology [RH06].

Ontologie-Konstrukte

Die heute gängigen Sprachen *Ressource Description Framework (RDF)* und *Web Ontology Language (OWL)* zum Aufbau von Ontologien nutzen und erweitern sich gegenseitig oder schränken sich zum Teil auch ein. Bevor die Ontologie-Sprachen im nachfolgenden Abschnitt konkreter mit ihrer Syntax/Serialisierung erläutert werden, sollen an dieser Stelle zunächst die grundlegenden Konstrukte eingeführt werden, auf denen sie basieren.

Als Basis dient die Annahme, dass sich die Fakten, Aussagen, Zusammenhänge und Regeln mithilfe einer Menge von Knoten und gerichteten Kanten zwischen den Knoten darstellen lassen. Der dadurch entstehende Graph wird mithilfe von Tripeln aus Subjekt, Prädikat und Objekt formalisiert. Ein solches Tripel beschreibt den Anfang (Subjekt) und den Endpunkt (Objekt) einer Kante sowie die Bezeichnung der Kante selbst (Prädikat). Analog zur objektorientierten Programmierung wird bei Knoten zwischen Individuen/Instanzen (Individuals) und ihren übergeordneten Typen/Klassen (Classes) unterschieden. Zusätzlich kann es sich bei einem Endknoten noch um ein Literal, also einen konkreten Wert in Form einer Zahl oder einer Zeichenkette handeln. Mithilfe von Merkmalen (Properties) werden die Beziehungen (Kanten) zwischen Knoten ausgedrückt. Merkmale, die auf ein Literal

führen, werden als Datenmerkmale (Datatype-Properties) bezeichnet, Objektmerkmale (Object-Properties) verknüpfen zwei Individuen miteinander [HKR+08]. Will man z. B. ausdrücken, dass ein Lösungselement `Delta-Roboter-Antrieb` existiert, das aus dem Servomotor `AM8023` und dem Servoverstärker `AX5103` besteht und ein Nennmoment von 1,0 N m hat, können die in Bild 2-12 in abstrakter Syntax dargestellten Tripel[10] verwendet werden. Sie führen zu dem ebenfalls abstrakt gezeigten Graphen.

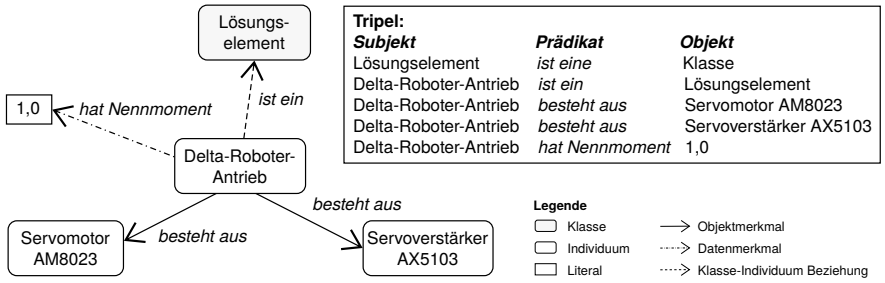


Bild 2-12: Ontologie-Konstrukte am Beispiel (abstrakte Syntax)

Die konkrete Syntax der spezifizierten Ontologie-Sprachen wird im nächsten Abschnitt eingeführt. Zur Darstellung der Graphen gibt es jedoch keine einheitlich festgelegte Symbolik. In dieser Arbeit werden daher im Folgenden die Analogien zur objektorientierten Softwareentwicklung ausgenutzt, um die etablierten und semantisch definierten UML-Klassendiagramme/SysML-Blockdefinitionsdiagramme verwenden zu können (vgl. Abschnitt 2.3.2). Ein wesentlicher Unterschied zu diesen Sprachen ist jedoch, dass Klassen sich in einer Ontologie nicht gegenseitig ausschließen. Eine Instanz kann hier mehreren Klassen zugehörig sein und von beiden erben. Dieser Umstand macht im Übrigen auch ein wesentliches Unterscheidungsmerkmal zwischen Ontologien und Taxonomien aus [Den12]. Um Fehlinterpretationen vorzubeugen, wird daher im Folgenden, der Terminologie des *World Wide Web Consortium (W3C)* folgend, nur noch von Individuen und nicht von Instanzen gesprochen.

Ontologie-Sprachen

Grundvoraussetzung für die Interoperabilität im Semantic Web ist die Einhaltung einheitlicher Sprachstandards zur Formulierung von Wissensmodellen. Für den Aufbau von Ontologien ist eine Reihe verschiedener Sprachen entwickelt worden,

[10] Insbesondere beim letzten Tripel wird offensichtlich, dass die Begriffe Subjekt, Prädikat und Objekt hier nicht streng äquivalent zur Grammatik der deutschen Sprache zu verstehen sind.

die weitere Untervarianten besitzen und sich in ihrer Ausdrucksstärke zum Teil deutlich unterscheiden. Sie sind spezifiziert durch sog. Recommendations des W3C. Im Folgenden werden die beiden Hauptvertreter RDF(S) (Ressource Description Framework) und OWL sowie deren Grundlage, die *Extensible Markup Language (XML)*, vorgestellt. Eine umfassendere Beschreibung findet sich u. a. in [HKR+08].

Extensible Markup Language (XML): Als Basis zur Annotation von Metadaten dient die XML, welche durch die entsprechende W3C Recommendation spezifiziert wird [BPS+08]. Der Begriff „Markup“ steht für das „Etikettieren“ von Textteilen in einem Dokument. Das Wort „Extensible“ verdeutlicht, dass es sich bei XML um eine Meta-Sprache handelt, die dazu genutzt wird, andere Markup-Sprachen zu definieren [HKR+08]. Dies geschieht durch strukturelle und inhaltlich Einschränkung, d. h. Festlegung auf anwendungsspezifische Tags mithilfe des spezifischen XML-Schemas[11]. Der bekannteste Vertreter für ein XML-Derivat ist (X)HTML, das zum Gestalten von Webseiten verwendet wird. Zum Beispiel wird mit dessen Hilfe in der folgenden Codezeile durch den „-Tag“ gängigen Web-Browsern mitgeteilt, dass ein Teil des Texts fett anzuzeigen ist (nämlich der zwischen und). Das <a>-Element kennzeichnet einen Link. Die zugehörige Ausgabe im Browser-Fenster ist direkt darunter dargestellt.

```
<b>Lösungselemente</b> gibt es <a href="http://wwwhni.upb.de/">hier</a>.
```

Lösungselemente gibt es hier.

Im Allgemeinen legt man XML-Elemente fest, welche die abzuspeichernden Informationen und Daten in einen logischen Zusammenhang bringen. Die Struktur der Daten wird durch die Verschachtelung der Elemente (Tags) zu einem Baum. Daneben können sog. Attribute Informationen über ein bestimmtes Element ausdrücken, wie das Ziel des Links im obigen Beispiel (href = Hyper(text)-Referenz). Welche Elemente und Attribute zur Verfügung stehen und ihre Restriktionen werden anwendungsspezifisch durch das XML-Schema festgelegt. Dabei kann man jedoch auf einen enormen Fundus an bestehendem XML-Vokabular und vielerlei Werkzeuge zurückgreifen. Letztere sind, wohlgeformte XML-Daten vorausgesetzt, zum Beispiel in der Lage, den Baum nach Informationen zu durchsuchen (engl. to parse) und/oder ihn umzustrukturieren [ST11, HKR+08].

Ziel des W3C ist es, die Wiederverwendung von Dokumenten zu ermöglichen, die mithilfe von XML spezifiziert wurden. Damit es zu keiner Verwechslung der Elemente unterschiedlichen Ursprungs kommen kann, müssen die Elementnamen

[11] Neben dem neueren XML-Schema Standard finden auch noch sog. DTDs (Document Type Definitions) Verwendung.

eindeutig sein. Dies wird insbesondere dann wichtig, wenn man an die Nutzung der XML zur Definition von Ontologie-Sprachen denkt, welche dazu dienen sollen, verteilte Inhalte im Semantic Web zu beschreiben. Zu diesem Zweck wurden die sogenannten *Uniform Resource Identifier (URI)* eingeführt [BHL+09]. URIs sind eine im Grunde beliebige Kette aus US-ASCII-Zeichen[12]. Allerdings wird in aller Regel ein *Uniform Resource Locator (URL)*, also die bekannte Form der Internetadressen verwendet und dem Elementnamen vorangestellt. Das bedeutet jedoch nicht, dass sich dahinter auch tatsächlich ein abrufbarer Inhalt im WWW befinden muss. Zusammen mit den möglichen XML-Elementen und Attributen bilden URIs den sogenannten Namensraum (engl. namespace). Um nicht jedes Mal die komplette URI schreiben zu müssen und um die Übersichtlichkeit zu verbessern, werden zu Beginn eines XML-Dokuments Abkürzungen (prefix) eingeführt, die in diesem Gültigkeit besitzen [ST11]. Die Tabelle 2-1 zeigt beispielhaft gängige Abkürzungen aus dem Semantic-Web-Umfeld.

Tabelle 2-1: Gängige Abkürzungen für Namensräume

Abk.	Uniform Resource Identifier	Erläuterung
xml:	http://www.w3.org/XML/1998/namespace#	XML-Basisnamensraum. Die Abkürzung xml: ist eindeutig festgelegt [BHL+09].
xsd:	http://www.w3.org/2001/XMLSchema#	Der XML-Schema-Namensraum beinhaltet u. a. verschiedene Datentypen
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#	Konstrukte der RDF-Sprache
rdfs:	http://www.w3.org/2000/01/rdf-schema#	Sprachbausteine aus RDF-Schema
owl:	http://www.w3.org/2002/07/owl#	OWL-Sprachbausteine zur Erweiterung von RDF und RDFS
swrl:	http://www.w3.org/2003/11/swrl#	Elemente für die Regelsprache SWRL

Ressource Description Framework (RDF(S)): Die XML bietet die Basis zur Definition von Ontologie-Sprachen, da mit ihrer Hilfe Maschinenlesbarkeit erreicht werden kann. Allerdings können die abgebildeten Informationen unterschiedlicher Quellen mit XML allein noch nicht in Verbindung gebracht werden. RDF(S) (vgl. [SR14, BG04]) bietet Möglichkeiten, verschiedene XML-Elemente in einen Zusammenhang zu bringen und dafür zu sorgen, dass bei einem Datenaustausch die ursprüngliche Bedeutung nicht verloren geht. Die erste Spezifikation wurde bereits vor dem Aufkommen des Semantic Web veröffentlicht [LS99], dennoch

[12] Die *Internationalized Resource Identifier (IRI)* sind eine Erweiterung und erlauben auch andere Zeichensysteme.

wurde RDF zur grundlegenden Semantic Web Sprache auserkoren auf der weitere, ausdrucksstärkere aufbauen sollten [HKR+08]. Dementsprechend bietet RDF elementare Ausdrucksmittel, um den erwähnten semantischen Graphen mittels Tripeln zu beschreiben und so ein Geflecht aus semantischen Beziehungen zu bilden [Bir06]. Diese beschränken sich im Wesentlichen auf Aussagen über Individuen, weshalb RDF folglich vor allem zur Befüllung der A-Box einer Wissensbasis dient. Um terminologisches Wissen zu übergeordneten Klassen beschreiben zu können, schlägt das W3C RDF-Schema (RDFS) vor [BG14]. Die beiden Namensräume werden heute allerdings nur noch aus historischen Gründen unterschieden.

Tabelle 2-2 zeigt einige wichtige Konstrukte von RDF(S). Die darin dargestellte Syntax entspricht der sogenannten Turtle Syntax [BB11]. Turtle ist eine alternative Serialisierung[13] der eigentlichen XML-Syntax für RDF (RDF/XML) [Bec04]. Hierdurch wird der Code deutlich besser lesbar, da die typischen Verschachtelungen in XML-Dokumenten aufgehoben werden. Auch sind die einzelnen Tripel deutlich besser zu erkennen. Wenngleich heute bereits viele Werkzeuge ebenso in der Lage sind, Turtle zu interpretieren, bleibt RDF/XML die Basis, in welche die Informationen überführbar sein müssen. In dieser Arbeit wird aufgrund der genannten Vorteile ausschließlich die Turtle-Syntax verwendet und im Folgenden anhand eines Minimalbeispiels (siehe Bild 2-13 auf Seite 43) schrittweise weiter eingeführt.[14]

Tabelle 2-2: Wichtige RDF(S)-Konstrukte [SR14]

Konstrukt	Syntax	Aussage
Class	**C** `rdf:type rdfs:Class.`	**C** ist eine RDF-Klasse.
Property	**P** `rdf:type rdf:Property.`	**P** ist ein RDF-Merkmal (Property).
type	**I** `rdf:type` **C**.	**I** ist ein Individuum (Individual) der Klasse **C**.
subClassOf	**C1** `rdfs:subClassOf` **C2**.	**C1** ist eine Unterklasse von **C2**.
subPropertyOf	**P1** `rdfs:subPropertyOf` **P2**.	**P1** ist ein Untermerkmal von **P2**.
domain	**P** `rdfs:domain` **C**.	Die Domäne von **P** ist die Klasse **C**, d. h. das Merkmal **P** beschreibt Individuen der Klasse **C**.
range	**P** `rdfs:range` **C**.	Der Wertevorrat von **P** sind die Individuen der Klasse **C**.

In diesem Beispiel werden zusätzlich zu den in Tabelle 2-1 aufgeführten Namensräumen die Präfixe `se:` für terminologisches Wissen über Lösungselemente

[13] Mit dem Wort Serialisierung bezeichnet man in der Informatik eine Umformulierung strukturierter Daten in eine sequentielle Form.

[14] Eine weitere benutzerfreundliche Syntax für OWL ist die *Manchester Syntax* [HP12]

(T-Box), `sea:` für spezielle Lösungselemente (A-Box) und `spa:` für Lösungsmuster-Individuen angelegt, um sie später als Abkürzung verwenden zu können. Die Definition in Turtle-Syntax könnte z. B. gemäß Quellcode 2-1 erfolgen (Zeilen 1–4). Des Weiteren werden in dem hier gezeigten elementaren RDF-Wissensmodell die Konstrukte `rdf:type` und `rdfs:Class` verwendet, um je eine übergeordnete Klasse für Lösungselemente (engl. solution element) und Lösungsmuster (engl. solution pattern) zu definieren (Zeilen 11 und 12) sowie das konkrete Lösungselement `DeltaRoboterAntrieb` als Individuum der entsprechenden Klasse anzulegen (Zeile 14). Dies erfolgt jeweils durch Tripel aus Subjekt, Prädikat und Objekt, welche durch einen Punkt abgeschlossen werden.

Quellcode 2-1: Beispiel eines elementaren Wissensmodells in RDF(S) mit Turtle-Syntax

```
1 @prefix se: <http://entime.uni-paderborn.de/ontology/
      SolutionElement#> .
2 @prefix sea: <http://entime.uni-paderborn.de/SolutionElements/
      SolutionElementABox#> .
3 @prefix sp: <http://entime.uni-paderborn.de/ontology/
      SolutionPattern#> .
4 @prefix spa: <http://entime.uni-paderborn.de/SolutionPatterns/
      SolutionPatternABox#> .
5 ...
6 ############## Wissensmodell ################
7 # Tripel aus:
8 # Subjekt----------Prädikat---Objekt
9 ##########################################
10 # T-Box
11 se:SolutionElement   rdf:type   rdfs:Class.
12 sp:SolutionPattern   rdf:type   rdfs:Class.
13 # A-Box
14 sea:DeltaRoboterAntrieb rdf:type se:SolutionElement.
```

Web Ontology Language (OWL): Die Web Ontology Language (OWL)[15] bedient sich der Klassen und Merkmale aus RDF(S) und führt weitere Konstrukte ein, um diese in komplexere logische Beziehungen zueinander setzen zu können [HKR+08, HKP+12]. Die Ausdrucksstärke wird dadurch größer. An dieser Stelle sieht man sich jedoch einem Zielkonflikt gegenübergestellt. Erhöht man die Ausdrucksmächtigkeit, so können die relevanten Konzepte zwar mit immer größerer Informationstiefe dargestellt werden. Andererseits sollen Inferenzmaschinen jedoch weiterhin in der Lage sein, implizite Beziehungen eindeutig herzuleiten, die Ontologie muss „entscheidbar“ bleiben [Den12]. Aus diesem Grund wurden die drei Varianten OWL-Full, OWL-DL und OWL-Lite definiert. Im Kontext dieser

[15] Der Ursprung des Buchstabendrehers beim Akronym OWL ist nicht eindeutig geklärt. Ein Grund ist, dass so die Aussprache vereinfacht wird.

Arbeit wird allerdings lediglich OWL-DL (Description Logic)[16] verwendet und vorgestellt, da sie die größte Ausdruckskraft bei gleichzeitiger Entscheidbarkeit garantiert [HKP+12]. Die wichtigste Einschränkung, die damit einhergeht, ist die Typeneindeutigkeit. Eine Entität kann demzufolge in OWL-DL niemals gleichzeitig eine Klasse und ein Individuum sein. Mithilfe von OWL können im Vergleich zu RDF u. a. Eigenschaften besser eingeschränkt, logische Zusammenhänge umfassender beschrieben und Äquivalenzen zwischen Entitäten unterschiedlichen Ursprungs definiert werden. Es können durch Schnittmengen, Vereinigungen, Kardinalitäten und Aufzählungen neue Klassen konstruiert werden [Bir06]. Die Tabelle 2-3 gibt eine Übersicht über wichtige OWL-Konstrukte.

Einige dieser Sprachbausteine seien im Folgenden am Beispiel erläutert. Bild 2-13 stellt dazu eine Erweiterung des RDF-Wissensmodells aus Quellcode 2-1 mittels OWL dar. Hierin ist sowohl die textuelle als auch die in dieser Arbeit verwendete, zugehörige graphische Darstellung gezeigt. Letztere nutzt Analogien zur UML bzw. SysML (vgl. Abschnitt 2.3.2). Die abkürzende Turtle-Syntax im linken Teil des Bildes sieht vor, dass wenn mehrere aufeinanderfolgende Tripel mit demselben Subjekt beginnen, dieses ab dem zweiten Mal weggelassen werden kann, sofern die Tripel mit einem Semikolon abgeschlossen werden. Schließt man ein Tripel mit einem Komma ab, so bedeutet das, dass beim darauffolgenden Tripel sowohl Subjekt als auch Prädikat identisch sind und weggelassen werden. In Zeile 2 wird wiederum zunächst die Klasse `SolutionElement` angelegt, dazu wird nun jedoch `owl:Class` verwendet. Der Punkt am Ende der Zeile definiert wie zuvor das Ende der Zuweisungen bzgl. dieser Klasse. Anschließend wird das Datenmerkmal `SolutionElementName` definiert, welches von Individuen der Klasse `SolutionElement` auf eine Zeichenkette (String) zeigt (Zeilen 4–6). OWL bedient sich dafür der `domain`- und `range`-Konstrukte aus RDF (vgl. Tabelle 2-2). Ab Zeile 8 werden die Eigenschaften des Objektmerkmals `hasSolutionPattern` modelliert. Dieses verbindet Individuen der Klasse `SolutionElement` mit denen der Klasse `SolutionPattern` (Lösungsmuster) und ist ein inverses Merkmal zu `abstractsSE`. Im letzten Abschnitt (ab Zeile 13) wird ein Individuum der Klasse `SolutionElement` angelegt. Diesem wird das Merkmal `producerName` mit dem Wert „Beckhoff Automation“ zugewiesen. Außerdem wird es über die zuvor definierte `hasSolutionPattern` Beziehung mit dem Lösungsmuster `Servodrive` verknüpft. Zuletzt wird mithilfe des Merkmals `comprisesSolutionElement` beschrieben, aus welchen Bestandteilen das Individuum `DeltaRoboterAntrieb` besteht.

Semantic Web Rule Language (SWRL): Die Semantic Web Rule Language (SWRL) wurde konzipiert, um noch ausdrucksstärkere Regeln definieren zu können.

[16] Die Bezeichnung *Description Logic* verdeutlicht den Zusammenhang mit der sog. Beschreibungslogik oder auch Prädikatenlogik erster Stufe, einem Teilgebiet der mathematischen Logik [HKR+08, Rau08].

Tabelle 2-3: Auflistung einiger wichtiger OWL-Konstrukte

Konstrukt	Syntax	Aussage
Class	**C** `rdf:type owl:Class.`	**C** ist eine OWL-Klasse und damit eine einschränkende Unterklasse von rdfs:Class (bei OWL-DL).
disjointWith	**C1** `owl:disjointWith` **C2**	Ein Individuum der Klasse **C1** kann nicht Individuum der Klasse **C2** sein.
equivalentClass	**C1** `owl:equivalentClass` **C2**	Die Klasse **C1** ist äquivalent zur Klasse **C2**.
NamedIndividual	**I** `rdf:type owl:NamedIndividual.`	**I** ist ein Individuum.
DatatypeProperty	**P** `rdf:type owl:DatatypeProperty.`	**P** ist ein Datenmerkmal.
ObjectProperty	**P** `rdf:type owl:ObjectProperty.`	**P** ist ein Objektmerkmal.
inverseOf	**P1** `owl:inverseOf` **P2**.	**P1** beschreibt im Vergleich zu **P2** die in die gegensätzliche Richtung gerichtete Kante.
FunctionalProperty	**P** `rdf:type owl:FunctionalProperty.`	**P** ist ein funktionales Merkmal. Die gerichtete Verbindung zwischen zwei Individuen über **P** ist eindeutig.
SymmetricProperty	**P** `rdf:type owl:SymmetricProperty.`	**P** ist ein symmetrisches Merkmal. Wenn Individuum **I1** mit Individuum **I2** verbunden ist, so gilt der gerichtete Zusammenhang auch in die gegensätzliche Richtung.
TransitiveProperty	**P** `rdf:type owl:TransitiveProperty.`	**P** ist ein transitives Merkmal. D.h. wenn die Individuen **I1** und **I2** genauso wie **I2** und **I3** über **P** verbunden sind, dann gilt das auch für **I1** und **I3**.

Sie ist (noch) kein offizieller Standard, sondern bisher nur ein Mitgliedervorschlag der W3C [HPB+04]. Die verwendete (visuell-lesbare) Syntax einer SWRL-Regel ist in Bild 2-13 (unten) zu sehen. Auf der linken Seite steht eine mit UND ($\wedge$) zusammengesetzte Bedingung, welche zur Schlussfolgerung rechts des $\rightarrow$ Operators führt. Im konkreten Fall muss es drei Individuen der Klasse `SolutionElement` geben, die mit den Variablen `?se1` bis `?se3`[17] bezeichnet werden. Für diese muss zusätzlich gelten, dass das Individuum `?se3` jeweils mit den anderen beiden über das Merkmal `comprisesSolutionElement` verbunden ist. Ist dies der Fall, kann darauf geschlossen werden, dass zwischen `?se1` und `?se2` die Beziehung `isCombineable` gilt. In natürlicher Sprache bedeutet das: Existieren zwei Lösungselemente, die beide Bestandteil eines dritten sind, so können sie immer kombiniert werden.

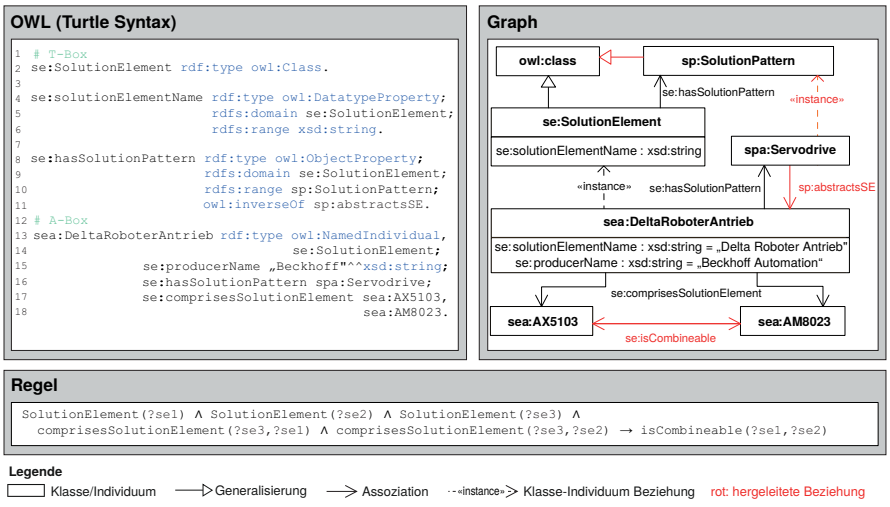


Bild 2-13: Beispiel zur verwendeten textuellen und graphischen Syntax

2.4.4 Inferenz und Abfrage von Wissen

Auf Basis des modellierten Wissens können logische Schlussfolgerungen (Inferenzen) gezogen und Anfragen an die Wissensbasis beantwortet werden. Beides wird in den nachfolgenden Abschnitten kurz erläutert.

[17] Durch das Fragezeichen wird gekennzeichnet, dass es sich um einen Platzhalter für Entitäten handelt.

Inferenzmaschinen

Inferenzmechanismen sind Regeln und Vorschriften, die es ermöglichen, Wissen herzuleiten. Die Aufgabe der Schlussfolgerung übernehmen sogenannte *Reasoner* oder *Inferenzmaschinen*, die durch Auswertung der Regeln und logischen Beziehungen neue Verknüpfungen und Aussagen berechnen. Folgende Schritte sind dazu nötig (vgl. [Den12]):

- Im Zuge der **Konsistenzprüfung** wird überprüft, ob die Ontologie logische Widersprüche enthält.
- Es wird geprüft, ob die modellierten Klassen Individuen besitzen können. Wurden Individuen definiert, obwohl sie logisch nicht möglich sind, ist die **Konzepterfüllbarkeit** nicht gegeben und die Wissensbasis inkonsistent.
- Bei der **Klassifikation** werden alle Klassenbeziehungen berechnet, sodass eine vollständige Klassenhierarchie entsteht.
- Zuletzt werden zu jedem Individuum die speziellsten Klassenzugehörigkeiten ermittelt. Man spricht von **Realisierung** zur Bestimmung der Typen.

Für OWL sind nur relativ wenige Inferenzmaschinen verfügbar. Eine der verbreitetsten ist *Pellet*, da sie zum einen den kompletten Umfang von OWL 2-DL sowie SWRL unterstützt und zum anderen online frei verfügbar ist [SPC+07]. Daneben gibt es z. B. *FaCT++* [TH06], *HermiT* [GHM+14] und *RACER* [RAC12]. Gegenüberstellungen der verschiedenen Eigenschaften und Leistungsfähigkeiten finden sich in [DCT+11] und [Abb12]. Alle Inferenzmaschinen für OWL basieren standardmäßig auf der sog. *Open World Assumption (OWA)*. Im Gegensatz zur *Closed World Assumption (CWA)* geht man bei der OWA davon aus, dass aus dem Fehlen eines Fakts im Datenbestand nicht automatisch darauf geschlossen werden kann, dass dieser nicht gilt. Einerseits ist die Annahme nachvollziehbar, wenn man die Anwendung im Semantic Web vor Augen hat, bei der das Wissensmodell ständig dezentral erweitert wird. Andererseits kann sie das Reasoning komplizierter machen, denn eine Schlussfolgerung kann solange nicht gezogen werden, wie keine definitive Aussage dazu enthalten ist. Damit geht einher, dass die sog. *Unique Name Assumption (UNA)* normalerweise nicht getroffen wird. Das bedeutet, dass zwei unterschiedlich bezeichnete Entitäten allein aufgrund dieser Tatsache nicht automatisch auch tatsächlich verschieden sind.

Im Falle des kleinen Beispiels aus Bild 2-13 können die im Graph der Ontologie rot eingezeichneten Verbindungen automatisch hergeleitet werden. Die Beziehung `isCombineable` folgt, wie bereits erläutert, aus der formulierten SWRL-Regel. Da `hasSolutionPattern` zuvor als inverses Merkmal zu `abstractsSE` definiert wurde (Bild 2-13 Zeile 11), kann letzteres Merkmal zwischen den Individuen `Servodrive` und `DeltaRoboterAntrieb` inferiert werden. `Servodrive`

muss vom Typ `SolutionPattern` sein, da diese Klasse als `range` des Merkmals `hasSolutionPattern` angegeben wurde (Zeile 10).

Abfrage von semantisch aufbereitetem Wissen

Bisher wurden Mittel zur Erzeugung einer Wissensbasis dargestellt. Es fehlen nun noch Wege, den Zugriff auf das Wissen zu ermöglichen, z. B. mithilfe einer Benutzerschnittstelle (vgl. Bild 2-11 auf Seite 31). Es bedarf Ausdrucksmittel, um eine Anfrage an die Wissensbasis stellen zu können. Für RDF wurde hierzu die Abfragesprache *SPARQL* (rekursives Akronym für: **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage) entwickelt [HS13]. Sie orientiert sich in Art und Syntax an dem für Datenbanken schon lange bekannten Standard *SQL* [ISO11].

Quellcode 2-2 zeigt ein Beispiel einer SPARQL-Anfrage, die Wissen aus der Beispielontologie (Bild 2-13) abfragt. Nach der Einführung der genutzten Namensraumpräfixe (Zeilen 1–5) folgt der eigentliche Frageteil, beginnend mit dem Schlüsselwort `SELECT`. Es werden Variablen eingeführt, die Bestandteil der Antwort sein sollen und mit einem Fragezeichen beginnen. Hier werden sie mit `?se` und `?seComponent` bezeichnet. Im `WHERE`-Teil werden anschließend, analog zur Turtle-Syntax, Tripel unter Nutzung dieser Variablen formuliert. Das Ergebnis der Anfrage sind alle Entitäten, die anstelle der Variablen eingesetzt werden können. In diesem Beispiel sind das alle Entitäten der Wissensbasis vom Typ `SolutionElement`, die den `producerName` „Beckhoff Automation“ haben und mit dem Individuum `Servodrive` über das Merkmal `hasSolutionPattern` verbunden sind. Ist eines dieser Kriterien nicht erfüllt, ist die Entität nicht Bestandteil der Antwort. Soll ein Tripel keinen ausschließenden Charakter haben, so muss es im `OPTIONAL`-Teil der Anfrage definiert werden (Zeilen 11–13).

Quellcode 2-2: Beispiel einer SPARQL-Anfrage

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4  PREFIX se: <http://entime.uni-paderborn.de/ontology/
       SolutionElement#>
5  PREFIX spa: <http://entime.uni-paderborn.de/SolutionPatterns/
       ActuatorSolutionPatterns#>
6  SELECT ?se ?seComponent
7  WHERE {
8      ?se rdf:type se:SolutionElement .
9      ?se se:producerName "Beckhoff Automation"^^xsd:string .
10     ?se se:hasSolutionElement spa:Servodrive .
11     OPTIONAL{
12          ?se se:comprisesSolutionElement ?seComponent .
13          ?seComponent rdf:type se:SolutionElement . }
14 }
```

Wird diese Anfrage an die Wissensbasis in Bild 2-13 gestellt, ist das Individuum `DeltaRoboterAntrieb` die einzige Lösung für `?se`. Zusätzlich werden `AX5103` und `AM8023` als `?seComponent` ausgegeben, da sie über `comprisesSolutionElement` mit `DeltaRoboterAntrieb` verknüpft sind und die entsprechenden Tripel im `OPTIONAL`-Teil der Anfrage stehen. Dieses Ergebnis kann zum Beispiel als XML serialisiert und ausgegeben werden. Über die Abfrage von Informationen hinaus besteht auch die Möglichkeit, neue Tripel zu generieren oder die Wissensbasis zu modifizieren. Hierzu wird ein `CONSTRUCT`- bzw. ein `DELETE`- oder `INSERT`-Teil eingefügt [GPP13]. Da SPARQL ursprünglich für RDF entwickelt wurde, können OWL- und RDF(S)-Inhalte nicht vollständig abgefragt werden. Jedoch gibt es bisher kaum Alternativen von praktischer Bedeutung. Auf Basis von SWRL wurde in [OD09] die Anfragesprache SQWRL für OWL vorgeschlagen, die innerhalb des Ontologie-Editors Protégé nutzbar ist. Detaillierte Erläuterungen zu SPARQL im Allgemeinen finden sich zum Beispiel in [HKR+08].

2.4.5 Werkzeuge für das Semantic Web

Zum Umgang mit den zuvor beschriebenen Sprachen sind verschiedene Werkzeuge verfügbar, einige davon auch frei zugänglich. An dieser Stelle soll ein kurzer Überblick gegeben werden, der den Fokus auf die genutzten Werkzeuge legt. Eine umfassendere Übersicht findet sich zum Beispiel in [WWH08]. Der bekannteste Ontologie-Editor ist Protégé[18] [KFN+04, GMF+03]. Mit ihm können zum einen RDF(S)- und OWL-Ontologien inkl. SWRL-Regeln erstellt werden. Zum anderen ermöglichen Plug-ins die Nutzung von Inferenzmaschinen und die Auswertung von Anfragen. Protégé wird entwickelt am *Stanford Center for Biomedical Informatics Research* der *Stanford University*, ist frei verfügbar und basiert wie viele andere Werkzeuge auf der sogenannten OWL API[19]. Hierbei handelt es sich um eine Java-Programmierschnittstelle, welche Parser für die gängigen Sprachserialisierungen und eine Schnittstelle für Reasoner zur Verfügung stellt.

Darüber hinaus sind diverse Systeme zur Auswertung großer Mengen von RDF-Daten (Triple Stores) frei verfügbar und auch kommerzielle Datenbanksysteme bieten zunehmend Unterstützung von RDF und OWL an [HKR+08]. Ein verbreiteter Triple Store ist *Fuseki*[20] als Nachfolger von *Joseki*. Fuseki erlaubt die Interaktion mit der Wissensbasis über SPARQL und HTTP. Fuseki ist Bestandteil des *Jena Framework*, welches darüber hinaus noch weitere Java-Entwicklerwerkzeuge und Systeme für den Zugriff, das Verwalten und das Manipulieren von semantischen Netzen bietet. Ziel ist die integrierte Implementierung aller W3C-Recommendations im Zusammenhang mit RDF [CDD+04].

[18] http://protege.stanford.edu/
[19] http://owlapi.sourceforge.net/
[20] http://jena.apache.org/documentation/serving_data/index.html

Letzter Bestandteil einer umfassenden Werkzeug-Landschaft für das Semantic Web sind die *Semantic Web Crawler* [Bid04]. Während konventionelle Web Crawler von Suchmaschinen wie Google regelmäßig HTML-Seiten nach neuen textuellen Inhalten durchforsten, indem sie sich über Verlinkungen von URL zu URL „hangeln“, muss diese Funktionalität für das Ziel des Semantic Web auf RDF- und OWL-Metadaten erweitert werden. Die verteilten, semantisch aufbereiteten Inhalte werden zu einem Wissensmodell kombiniert, um sie nutzbar zu machen [DB15]. Ein Framework für diesen Zweck bietet z. B. *Slug* [SN12]. Die Suchmaschine *SWOOGLE*[21] nutzt einen Semantic Web Crawler, um Semantic-Web-Dokumente zu indizieren und sie so sowohl für den Menschen als auch für Maschinen auffindbar zu machen [FPC+04].

2.5 Verwandte Arbeiten

Nachfolgend werden, sortiert nach Themenschwerpunkten, einige verwandte Arbeiten diskutiert. Auf Grundlage der Beschreibung und des bereits skizzierten Stands der Wissenschaft und Technik erfolgt jeweils anschließend eine kurze Bewertung.

2.5.1 Wiederverwendung von Lösungswissen

In den beteiligten Disziplinen der Entwicklung mechatronischer Systeme gibt es unterschiedlich ausgeprägte Bestrebungen zur Wiederverwendung spezifischen Lösungswissens. Der Abstraktionsgrad sinnvoll wiederverwendbarer Lösungen variiert dabei sowohl im Laufe des Prozesses, als auch zwischen den Disziplinen. Während bei der Informationsverarbeitung eines mechatronischen Systems, die in der Regel anwendungsspezifisch auszuprägen ist, normalerweise hauptsächlich Lösungsprinzipien wiederverwendbar sind, erlaubt der klassische Maschinenbau die Nutzung vergleichsweise sehr konkreter Teillösungen. Solch konkrete Lösungen werden unabhängig von der Disziplin, der gebräuchlichen Konvention folgend, im Kontext dieser Arbeit mit dem Begriff *Lösungselement* bezeichnet.

Wiederverwendung von Lösungselementen

Lösungselemente stellen eine konkrete, fertige Lösung dar, die zum Beispiel direkt bei einem Zulieferer bezogen werden kann und dementsprechend zum Teil auch als Zulieferkomponente oder Katalogteil bezeichnet wird. Sie sind vor allem in der Konstruktionstechnik sehr gebräuchlich. So stellen zum Beispiel die Konstruktionskataloge nach Roth ein viel genutztes Standardwerk des klassischen Maschinenbaus dar [Rot01]. Darüber hinaus definiert die Reihe DIN 4000 sog.

[21] http://swoogle.umbc.edu/

Normteile, deren CAD-Modelle u. a. in elektronischen Bibliotheken verfügbar sind und bei Neukonstruktionen herangezogen werden [DIN12]. Mit dem Begriff Lösungselement sind jedoch nicht nur einfache Zukaufteile wie Lager, Schrauben etc. gemeint. Vielmehr wird von der sogenannten Lösungselement-Kaskade ausgegangen, bei der mehrere Lösungselemente Teil eines höherwertigen Lösungselements sind. Bild 2-14 illustriert dies am Beispiel eines Wälzlagers, das Bestandteil eines Motors ist, welcher wiederum in einem Roboter verbaut ist.

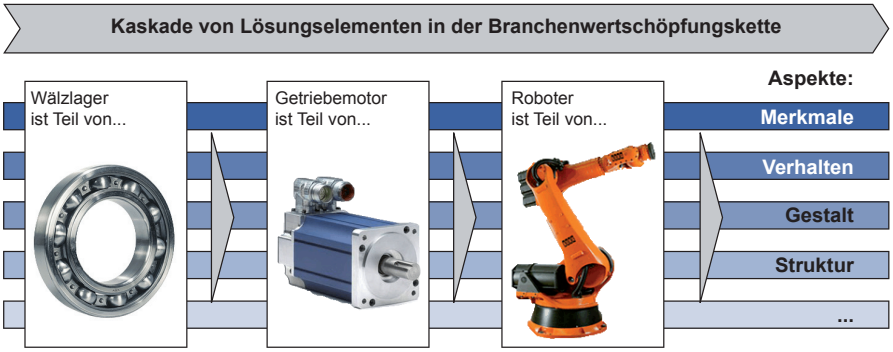


Bild 2-14: Kaskade von Lösungselementen [GTS14, S. 12]

Klassifikationsstandards wie z. B. eCl@ass [ECL09], ETIM [ETI11] oder UNSPSC [UNS14] definieren feste hierarchische Strukturierungen und z. T. auch eine Reihe von Aspekten, mit denen Lösungselemente beschrieben werden. Sie dienen in erster Linie dem elektronischen Geschäftsverkehr (E-Commerce). Der Austausch zwischen Zulieferer und Kunde soll vereinfacht werden, indem Ersterer seine Produkte gemäß des Standards klassifiziert, sodass sie u. a. besser auffindbar und vergleichbar sind. Die Durchdringung der Standards ist jedoch relativ gering. Laut einer repräsentativen Umfrage unter deutschen Unternehmen aus dem Jahre 2002 setzen lediglich rund 35 % von ihnen Standardproduktklassifikationen ein, mit 11,3 % am häufigsten wird eCl@ss verwendet [OBM02].

Die internationale Norm ISO 13584 beschreibt eine einheitliche Strukturierung von Teilebibliotheken (engl. Parts Library (PLIB)) zur Integration verschiedener Klassifikationssysteme und legt dazu ein Datenmodell fest [ISO01]. Das PLIB-Datenmodell sowie das ebenfalls spezifizierte XML-Austauschformat *ontoML* (ISO 13584-32:2010) ist Grundlage für eine Reihe weiterer Arbeiten. So basiert mittlerweile auch eCl@ss als einziger internationaler Klassifikationsstandard auf einem normenkonformen Datenmodell. Unter anderem erarbeitet Sander aufbauend auf dem PLIB-Datenmodell ein Konzept für eine digitale Bibliothek von Lösungselementen mit Fokus auf die Konstruktion [San00].

Wiederverwendung von Lösungs- und Entwurfsmustern

In der Softwaretechnik hat sich die Nutzung von Entwurfsmustern (engl. design patterns) zur Entwicklung objektorientierter Anwendungen bewährt [GHJ+94]. Diese beschreiben in erster Linie einen Lösungsweg für wiederkehrende Problemstellungen und enthalten Informationen über Name und Klassifikation, Zweck, Motivation für den Einsatz, Anwendbarkeit, Struktur, beteiligte Akteure/Klassen und deren Zusammenspiel, Vor- und Nachteile, Implementierungstipps, Praxiseinsatz und Querverweise, zumeist auch Beispielcode. Grundlage ist die allgemeine Definition eines Musters nach Alexander et al. [AIS+77]:

> *Jedes Muster beschreibt ein in unserer Umwelt immer wieder auftretendes Problem, beschreibt den Kern der Lösung dieses Problems, und zwar so, dass man diese Lösung millionenfach anwenden kann, ohne sich je zu wiederholen [AIS+95, S. 12].*

In der Konstruktionslehre ist der Begriff des *Wirkprinzips* etabliert. Ein Wirkprinzip bezeichnet nach Pahl/Beitz den Zusammenhang zwischen geometrischen und stofflichen Merkmalen (Wirkgeometrie, Wirkbewegung und Werkstoff) und stellt den Lösungsgedanken für eine Funktion dar [PBF+13]. Aufbauend auf den Arbeiten von Suhm [Suh93] sowie Lossack und Grabowski [LG00] u. a. erweitert Dumitrescu die Beschreibung eines Wirkprinzips zum sogenannten *Lösungsmuster*, das für alle Fachdisziplinen, die an der Entwicklung fortgeschrittener mechatronischer Systeme beteiligt sind, nutzbar ist [Dum10]. Diese Lösungsmuster (LM) unterstützen zusätzlich insbesondere den frühzeitigen Entwurf der Informationsverarbeitung und enthalten sechs verschiedene Aspekte (siehe Bild 2-15). Dabei entsprechen die Aspekte `Funktionen`, `Wirkstruktur` und `(Software-) Verhalten` im Wesentlichen den in CONSENS definierten Partialmodellen (siehe Abschnitt 2.3.2). Daneben werden `Merkmale` des Lösungsmusters inkl. des Wertebereichs aufgelistet, der `Kontext` mithilfe von Beispielen für den Einsatz dargestellt sowie das Prinzip der Lösung im Aspekt `Lösungsprinzipien` verdeutlicht.

Lösungsmuster stellen je nach Anwendungsfall eine geeignete Abstraktion einer Klasse von Lösungselementen dar und können selbst aus wiederum anderen Lösungsmustern bestehen (vgl. Bild 2-14) [GTS14]. Neben der Spezifikation eines Lösungsmusters wird in [Dum10] eine Lösungsmusterwissensbasis in Form zweier Datenbanken prototypisch umgesetzt. Anacker erarbeitet darauf aufbauend ein Instrumentarium zur Identifizierung, Dokumentation und Anwendung von Lösungsmustern, welches verallgemeinernd z. B. auch Anforderungs-, Systemarchitektur-, Systemanalyse- oder Geschäftsmodellmuster berücksichtigt [Ana15].

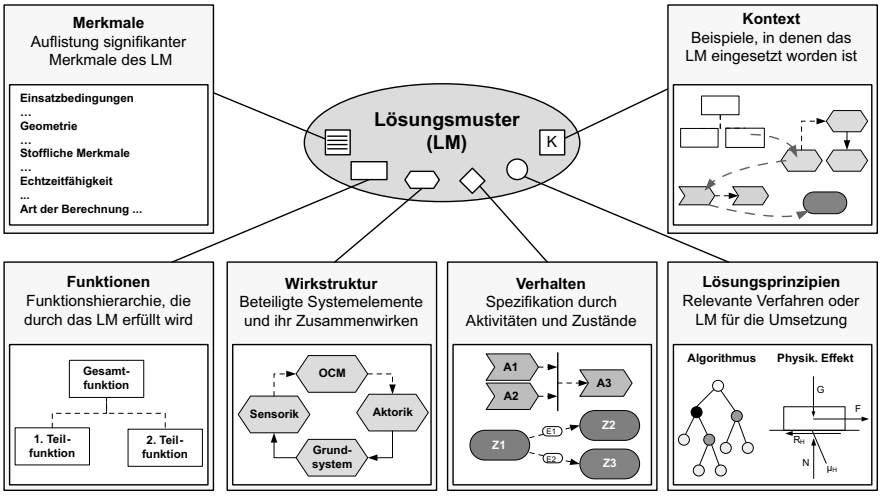


Bild 2-15: Spezifikation eines Lösungsmusters im Überblick [Dum10, S. 130]

Wiederverwendung von Dynamikmodellen

Um den Aufwand und die Fehleranfälligkeit bei der Modellierung des oftmals komplexen dynamischen Verhaltens mechatronischer Systeme zu reduzieren, ist es sinnvoll, Dynamikmodelle als Lösungswissen für die mechatronische Komposition (vgl. Abschnitt 2.2.2 auf Seite 18) anzusehen.

Mocko et al. entwickeln daher ein Konzept für ein Repository von Verhaltensmodellen und setzen es prototypisch um [MMP+04]. Um die Auswahl aus einem solchen zu ermöglichen, werden Metadaten über Annahmen, Grenzen, Genauigkeit und Kontext u. a. hinterlegt. Sie unterteilen das bei der Modellierung zum Einsatz kommende (Lösungs-) Wissen in zwei Kategorien: zum einen in das direkt im Modell formalisierte explizite Wissen, zum anderen in Metadaten, die Wissen (z. B. Annahmen, Grenzen) und Informationen (z. B. Autor, Version) über das Modell darstellen. Letztere würden jedoch oftmals nicht mit abgespeichert. Die Autoren geben darüber hinaus einen abstrakten, übergeordneten Ablauf für die Bereitstellung und Nutzung wiederverwendbarer Verhaltensmodelle an. Offen bleibt jedoch, wie das richtige Modell aus dem Repository ausgewählt werden kann [MMP+04].

In [BBT+98] beschreiben Breunese et al. eine generische Architektur für Bibliotheken von wiederverwendbaren Dynamikmodellen. Mithilfe einer allgemeinen Modellbeschreibungssprache werden die drei Sichten *Struktur* (technical component level), *physikalischer Effekt* (physical concept level) und *mathematische*

Beschreibung (mathematical level) eines Modells spezifiziert und gespeichert. Ein komplettes Modell kann anschließend durch Auswahl der geeigneten Modellfragmente aus jeder Sicht zusammengesetzt werden. Hierbei unterstützt die Bibliothek den Anwender durch eine Taxonomie/Klassifikation sowie durch Verknüpfungen zwischen den Modellfragmenten der verschiedenen Sichten.

Paredis et al. stellen einen Ansatz für ein Simulationsrahmenwerk vor. Dieses ermöglicht die Kombination rekonfigurierbarer, objektorientierter Verhaltensmodelle zu einem Gesamtmodell mithilfe von ungerichteten Port-Schnittstellen [PDS+01]. Es nutzt Informationen aus CAD-Modellen und eine Modellbibliothek. Auch hier bleibt die Auswahl des geeigneten Modells dem Benutzer überlassen.

Diese Auswahl wird im Kontext ereignisdiskreter Verhaltensmodelle von Brooks und Tobias adressiert. In [BT96] unterscheiden sie verschiedene Aspekte der Modellgüte wie Komplexität und Detaillierungsgrad. Weiterhin wird herausgearbeitet, dass vage Aussagen über diese beiden Kriterien zur Auswahl eines geeigneten Modells nicht ausreichen. Es bedürfe vielmehr klarer Richtlinien und quantifizierbarer Größen.

In [ONB02] werden allgemeine Probleme bei der Wiederverwendung von Simulationsmodellen identifiziert. Die Autoren fokussieren dabei jene, die über die generellen Schwierigkeiten bei der Wiederverwendung von Software-Code hinaus gehen. Zum Beispiel müssen Annahmen, Grenzen und Modellierungsziele jeder Komponente möglichst maschinenverständlich dokumentiert werden. Außerdem werden Modelle desselben Systems in unterschiedlichen Abstraktionsgraden benötigt. Aufgrund dieser Schwierigkeiten sei eine generalisierte, vollständig automatisierte Auswahl nicht zu erreichen. Vielmehr müsse es das Ziel sein, den Nutzer dabei zu unterstützen, mögliche Modelle aufzufinden, potentielle Inkonsistenzen zu erkennen und wiederverwendbare Komponenten aufzubauen. Die Forderung nach dem einfachsten Modell zur Lösung des Problems könne aus diesen Gründen nicht mehr ständig aufrecht erhalten werden und sei darüber hinaus aufgrund gestiegener Rechenkapazität nicht mehr zeitgemäß.

Robinson et al. stehen der Wiederverwendung kritisch gegenüber, wenn diese ungeplant erfolgt oder als Anpassung eines ähnlichen Modells verstanden wird, da dies die Einarbeitung in das bestehende Modell inkl. der Überprüfung der Richtigkeit von Annahmen etc. bedeutet. Hierdurch entstehe Aufwand, der den eines Neuaufbaus übersteigen kann [RNP+04].

Bewertung: Die effiziente Wiederverwendung von Lösungswissen ist bereits seit einiger Zeit Gegenstand der Forschung, jedoch besitzen die Ergebnisse zumeist nur disziplinspezifische Gültigkeit oder sind sehr stark von einer Disziplin geprägt. Standardklassifikationen und die Nutzung abstrakterer Muster zielen darauf ab, die extrem große Vielfalt am Markt verfügbarer Lösungen beherrschbarer zu machen.

Dennoch bleibt die Suche nach der geeigneten Lösung auch deshalb schwierig, weil Ansätze aus dem Umfeld des elektronischen Handels technische Kriterien zumeist nicht genügend berücksichtigen. Die flächendeckende Durchdringung ist bisher ebenfalls nicht gegeben, sodass sich der Anwender weiterhin mit uneinheitlichen Terminologien und Strukturierungen der Anbieter auseinandersetzen muss. Lösungsmuster ermöglichen es, Lösungswissen bereits früh in den Entwurfsprozess zu integrieren. Die Spezifikation von Dumitrescu ist hierfür besonders vielversprechend, da sie disziplinübergreifend ausgelegt ist. Jedoch berücksichtigt sie nicht den Aspekt der möglichst frühzeitigen Absicherung der Systemdynamik. Hierzu sind Simulationen mit idealisierten Modellen notwendig, deren Ergebnisse darüber hinaus die Lösungselement-Suche verbessern können, indem sie den Suchraum einschränken. Die Wiederverwendung von abgeschlossenen Dynamikmodellen für die Komponenten eines Systems ist sinnvoll und scheint prinzipiell möglich, besonders dann, wenn objektorientierte Modellierungssprachen und ungerichtete Port-Schnittstellen verwendet werden. Die Anpassung eines bestehenden Modells kann im Gegensatz dazu als in der Regel nicht zielführend angesehen werden (vgl. [RNP+04]). Darüber hinaus muss der Modellierer sowohl beim Auffinden als auch beim Aufbau wiederverwendbarer Dynamikmodelle unterstützt werden. Damit ein geeignetes, konsistentes und vollständiges Gesamtmodell entstehen kann, wird manueller Eingriff bzw. zumindest eine Modellüberprüfung durch den Anwender weiterhin nötig sein (vgl. [ONB02]).

2.5.2 Semantische Verknüpfung von Modellen

Um die Effizienz und Aussagekräftigkeit der verschiedenen Modelle im Entwurfsprozess mechatronischer Systeme sicherstellen zu können, muss deren Konsistenz gewährleistet werden. Dies betrifft u. a. die Struktur, Parameter und Detaillierungsgrade. Viele Ansätze beschäftigen sich daher mit der Schaffung einer konsistenzhaltenden, semantischen Verbindung der disziplinspezifischen Modelle mit einem disziplinübergreifenden Systemmodell. Hierbei kann zwischen föderativen und integrativen Ansätzen unterschieden werden.

Das SysML4Modelica Profil [RPC+12, PBB+10, JPB08] ermöglicht es, einen Teil des in SysML beschriebenen Systemmodells mithilfe der bidirektionalen SysML-Modelica-Transformation der INCOOSE nach Modelica zu übersetzen (basierend auf ModelicaML). Zu diesem Zweck werden Stereotypen für Modelica-Sprachkonstrukte definiert. Allerdings werden u. a. Informationen zur graphischen Darstellung nicht betrachtet. Laut [RPC+12] erweist sich darüber hinaus die Entwicklung des kompletten Dynamikmodells in SysML (integrativer Ansatz) und insbesondere die Beherrschung der verschiedenen Abstraktionslevel als nicht praktikabel bzw. unmöglich. Daher wird ein Vorschlag für einen gemischten,

iterativen Arbeitsablauf erarbeitet (föderativer Ansatz), welcher auf der formalen Beschreibung der Abhängigkeiten zwischen den Modellen gründet.

Die verbreitetste Technologie zur Transformation zwischen zwei Modellen (Modelltransformation), d. h. zur Beschreibung dieser Abhängigkeiten, ist die Triple Graph Grammar (TGG) [Sch94]. Hiermit wird ein Modell der Korrespondenzen zwischen zwei Modellen erstellt. Ein solches wird zum Beispiel von Gausemeier et al. genutzt um die Konsistenz zwischen dem disziplinübergreifenden Strukturmodell der Wirkstruktur und Komponentendiagrammen zur Beschreibung des Softwareverhaltens sicherzustellen [GSG+09]. Barbieri et al. [BFB14] nutzen TGGs für den Austausch zwischen disziplinspezifischen Modellen und einem zentralen SysML-Systemmodell. Die Methode wird an einem bereits existierenden Beispiel einer Abfüllanlage erarbeitet und getestet. Offen bleibt jedoch, welche Informationen der disziplinspezifischen Modelle übertragen werden müssen und in welcher Detaillierung. Darüber hinaus lassen sich mithilfe von TGG-Regeln nur syntaktische, nicht jedoch semantische Konsistenzbeziehungen beschreiben [GSG+09]. Die Verknüpfung von Modellen, die über heterogene Abstraktionsgrade sowie unterschiedliche Modellierungsziele und Ausdruckskraft verfügen, ist nicht ohne Weiteres möglich und daher Gegenstand aktueller Forschung [Rie14].

Qamar und Paredis nutzen SysML zur Modellierung von Abhängigkeiten in sechs verschiedenen Detailstufen [QP12]. Sie schlagen notwendige Erweiterungen der Sprachspezifikation vor und diskutieren den zu erwartenden, u. U. beträchtlichen Mehraufwand.

Chen et al. [CBP+09] stellen einen Ansatz für ein Rahmenwerk zur integrierten Entwicklung mechatronischer Systeme vor. Dieses basiert auf der komponentenweisen Modellierung, der Klassifikation und dem Austausch von Randbedingungen zwischen den Disziplinen Mechanik und Elektronik. Hierzu wird ein Netz aus Knoten und Kanten verwendet.

Mit der Software CATIA V6 wurde ein integrativer Ansatz umgesetzt: sie vereint verschiedene Werkzeuge, um die Schritte Requirements, Functional Design, Logical Design und Physical Design zu durchlaufen (vgl. Abschnitt 2.3). Der Informationsaustausch erfolgt über eine gemeinsame Datenbank [KK13].

Frühere Arbeiten von Koch [Koc05], Wittler [Wit02] u. a. beschäftigen sich ebenso mit der integrativen Modellierung und geben methodische Hinweise. Sie beschränken sich aber auf die integrative Modellierung von Gestalt und dynamischem Verhalten sowie auf spezielle Werkzeuge.

Ein weiterer Ansatz wird in [PPC+10] vorgestellt. Die Autoren stellen eine Möglichkeit zur integrativen Modellierung vor, beginnend mit den Anforderungen bis hin zum detaillierten Entwurf mithilfe von Dymola/Modelica. Dazu werden z. B. die Anforderungen an das System im Modelica-Text als Annotation gespeichert

und später automatisch in ein Diagramm überführt. Auch Funktionen werden als Block modelliert und über Modelica-Konnektoren verbunden.

Foeken und van Tooren diskutieren einen Modellgenerator für objektorientierte Simulationsmodelle (am Beispiel Modelica) mit dem übergeordneten Ziel einer automatisierten Generierung der Regelungs-/Steuerungsoftware mechatronischer Systeme [FT09]. Der vorgeschlagene Prozess startet bei einem Modell der Systemfunktionen und führt über ein System-/Strukturmodell sowie „qualitative Reasoning-Techniken“ zur benötigten Regelstrategie. Als einen Bestandteil erstellt der Modellgenerator mithilfe einer Wissensbasis ein geeignetes Streckenmodell zum Entwurf und zum Test der Software. Dazu wird zunächst eine Ontologie zur Repräsentation der Modelica-Sprache aufgebaut und um Konzepte und Beziehungen des Anwendungsbeispiels eines unbemannten Flugobjekts erweitert. Auf Basis der Schnittstellendefinition des Strukturmodells werden Komponentenmodelle ausgewählt, die benötigten Modellparameter sind jedoch nicht verfügbar [FV10].

Liang und Paredis beschreiben die ungerichteten Port-Schnittstellen von Simulationsmodellen in einer *port ontology* [LP03]. Diese beinhaltet neben den verschiedenen Ports auch Attribute bezogen auf ihre Form, Funktion und ihr Verhalten sowie Regeln für die Kompatibilität. Ziel ist es, eine automatisierte Komposition verschiedener Teilmodelle auf Basis der Systemarchitektur zu erreichen. Mithilfe der T-Box-Ontologie kann die Semantik von beliebigen Ports eindeutig und maschinenverständlich modelliert werden. Ihre Anwendung wird am Beispiel verschiedener Lego-Steine demonstriert.

Des Weiteren gibt es eine Reihe von Bestrebungen, standardisierte, toolunabhängige und disziplinübergreifende Austauschformate für Produktdaten und Modelle zu definieren. Beispiele hierfür sind OSLC und AutomationML. AutomationML ist ein auf XML aufbauendes Datenformat, das für die Speicherung und zum Austausch von Anlagenplanungsdaten gedacht ist. Hierzu werden die Aspekte *Struktur*, *Geometrie*, *Kinematik* und *Logik* einer Anlage beschrieben [AML14]. Das Ziel von OSLC ist es, Ressourcen über den kompletten Lebenszyklus des Systems werkzeugunabhängig zu verknüpfen und bearbeitbar zu machen. Zu diesem Zweck definiert OSLC sie als RDF-Merkmale [JS13]. Daneben existieren disziplinspezifische Formate wie STEP [DIN02] (vornehmlich, jedoch nicht ausschließlich zum Austausch von CAD-Daten genutzt) oder FMI [FMI14] zum Austausch von Dynamikmodellen.

Bewertung: Die Konsistenzhaltung der in Aussagekraft, Detaillierung und Modellierungszielen sehr unterschiedlichen Modelle im Entwurf mechatronischer Systeme ist eine Herausforderung, besonders dann, wenn föderative Ansätze gewählt werden. Diese nutzen im Gegensatz zu integrativen Ansätzen jedoch die speziellen Stärken der jeweiligen Werkzeuge, Sprachen und Formate. Außerdem ermöglichen födera-

tive Ansätze im Hinblick auf die Wiederverwendung von Lösungen die Integration bereits gespeicherter, wertvoller Entwicklungsartefakte aus den gewachsenen disziplinspezifischen Umgebungen. Als werkzeugunabhängiges Ausdrucksmittel sind Standards wie OSLC oder AutomationML entwickelt worden. Von den zahlreichen Formaten hat sich jedoch bisher keines branchenübergreifend durchgesetzt, wodurch der ursprüngliche Sinn eines Standards in gewisser Weise ad absurdum geführt wird. Vielversprechend erscheint die Nutzung eines disziplinübergreifenden Systemmodells als Mediator zwischen den disziplinspezifischen Modellen. Wenn dieses als Ontologie repräsentiert wird, können die Vorteile der semantischen Modellierung zum Beispiel beim „Übersetzen“ zwischen den Begriffswelten, bei der Kategorisierung oder beim Inferieren „neuen“ Wissens genutzt werden (vgl. [RP11]). Dieser Gedanke wird im folgenden Abschnitt aufgegriffen.

2.5.3 Ontologie-basierter Informationsaustausch

Für den ontologie-basierten Informationsaustausch zwischen verschiedenen Disziplinen, Organisationseinheiten, Standorten u. a. gibt es trotz der noch relativ jungen Technologie sehr viele Ansätze. In diesem Abschnitt sollen jedoch nur Ansätze thematisiert werden, die im skizzierten Umfeld intelligenter technischer Systeme Relevanz besitzen. Sie können unterteilt werden in solche, die auf den disziplinübergreifenden Austausch abzielen, und jene, die zur Unterstützung der Suche nach Lösungswissen gedacht sind.

Disziplinübergreifender Informationsaustausch

Das Potential von Ontologien zur Verknüpfung von Entwicklungsartefakten im Bereich der Ingenieurwissenschaften wird in [Kit06] diskutiert. Die Modellierung des Produktwissens mithilfe von Ontologien könne dazu dienen, Daten und Wissen auszutauschen, zu integrieren sowie zu kommunizieren und so dazu beitragen, auch implizites Wissen systematisch zu erfassen.

Das Knowledge Intensive Engineering Framework (KIEF) war eines der ersten Projekte, die die Nutzung dieses Potentials zum Ziel hatten [TUK96]. Werkzeugspezifische Modelle werden über einen auf mehreren Ontologien basierenden Metamodell-Formalismus verknüpft. Dazu werden die jeweiligen disziplinspezifischen Zusammenhänge sowie u. a. Beziehungen zwischen Entitäten, Attribute, physikalische Phänomene und Regeln modelliert [TUK96, Yos00, YUT+04]. Neben der funktionalen Sicht auf das System konzentrieren sich die Arbeiten vornehmlich auf Modelle aus den Disziplinen Mechanik und Dynamik (CAD-Modelle, FEM). Auch haben sich die Ontologie-Sprachen seit dieser Zeit deutlich weiterentwickelt. Nach Kitamura kann solch ein Mapping jedoch genutzt werden, um Wissen zu teilen [Kit06].

OntoSTEP ist ein auf STEP aufbauendes Format, das dieses um Semantik anreichert. Hierzu wurde ein Mapping nach OWL-DL erarbeitet mit dem Ziel, Modelle auf Validität und Konsistenz testen sowie Wissen inferieren zu können [KBF+09].

Silveira Mastella schlägt eine auf Ontologien und semantischer Annotation basierende Lösungsmöglichkeit zur Verknüpfung von unterschiedlichen Modellen der Ingenieurwissenschaften vor [Sil10]. Angewendet und entwickelt wird ein initiales „Framework“ am Beispiel geologischer Modelle für Erdölspeicher. Die Integration von Lösungswissen wird nicht thematisiert.

Zhan erarbeitet einen Ansatz zum ontologie-basierten Austausch von Informationen zwischen CAD- und Computer-Aided Engineering (CAE)-System über den Produktlebenszyklus [Zha07]. Die verwendeten Beispiele sind jedoch ausschließlich aus der Disziplin „Mechanik“ und wenig komplex. Zudem erfordert der Ansatz die Modellierung werkzeugspezifischer Semantik. Patil et al. verfolgen ein ähnliches, auf Mechanik fokussiertes Konzept [PDS05]. Die verwendete Sprache DAML+OIL ist jedoch mittlerweile veraltet und wird nicht mehr weiterentwickelt. Der Nachfolger OWL ermöglicht deutlich mehr Ausdruckskraft.

In [BJF08] werden verschiedene *Domain Ontologies* für die Disziplinen Mechanik, Dynamik, Hydraulik und Regelungs-/Softwaretechnik vorgeschlagen, die mithilfe der sog. „Mechatronic System Ontology“ verknüpft werden. Letztere beinhaltet allgemeine Konzepte und Beziehungen im Entwurfsprozess mechatronischer Systeme. Ein User-Interface erlaubt den Zugriff und die Wartung.

Park und Fishwick stellen eine integrative „Multimodell-Umgebung“ auf Basis von Ontologien vor. Hierbei stehen vor allem Grafiksimulationen im Mittelpunkt, weshalb die Blender Computerspiele-Grafiksoftware als elementarer Bestandteil gewählt wird. Die spätere Anbindung von Modellen der Systemdynamik, Petri-Netzen u. a. wird jedoch vorgesehen [PF05].

Graves verfolgt das Ziel, Entwickler dazu zu befähigen, die grafische Syntax von SysML zu nutzen und das Modell anschließend zur Analyse und Verifikation durch Reasoning-Tools zu exportieren. Dazu entwickelt er in einem ersten Schritt eine Übersetzung von SysML-Blockdiagrammen nach OWL, sodass Erstere zum Entwurf einer Systemwissensbasis genutzt werden können [Gra09].

Semantische Aufbereitung von Lösungswissen

Kitamura und Mizoguchi bereiten funktionales Wissen auf [KM03, MK01]. Die Grundlage für die Systematisierung von Wissen bilden zwei abstrakte Ontologien, *functional ontology* und *device ontology* genannt. Erstere unterscheidet Energie-, Objekt- und Informationsfunktionen sowie acht sog. „Meta-Funktionen“ (*ToProvide, ToDrive, ToEnable, ToAllow, ToPrevent, ToImprove, ToEnhance, ToControl*), welche den Zusammenhang zwischen ihnen bilden [MK01]. Das Ziel

der *device ontology* ist es, eine disziplinunabhängige Basis zu schaffen, um ein System mit seinen Systemelementen zu modellieren [KM03]. Systemelemente werden dazu als Black-Box mit Ein- und Ausgängen gesehen, die den Zustand eines durchfließenden Objekts beeinflusst. In [KM03] wird außerdem die Machbarkeit der Wiederverwendung von aufbereitetem funktionalen Lösungswissen gezeigt. Dieses kann dem Entwickler entsprechend seiner Sicht auf das System präsentiert werden.

Gaag untersucht den Zugriff auf existierende Lösungen aus Sicht der Funktionen unter Einsatz semantischer Technologien [Gaa10]. Hierzu wird ein Vorgehensmodell erarbeitet, das sowohl die Entwicklung der Ontologie als auch deren Einsatz zur Suche nach Prinziplösungen beinhaltet. Zwei Fallbeispiele zeigen das große Potential von semantischen Technologien zur Lösungsfindung. Die Bewertung der Eignung einer Lösung für die konkrete Aufgabenstellung bleibt hier jedoch dem Nutzer überlassen. Im Ausblick wird aufgezeigt, dass es notwendig ist, sowohl den hohen Aufwand bei der manuellen Befüllung der Ontologie als auch die Lösungsalternativen zu reduzieren. Letzteres soll durch zusätzliche Kriterien erreicht werden, denn die alleinige Betrachtung der Funktion könne nur eine grobe Einschränkung des Lösungsraums geben [Gaa10].

Yang et al. entwickeln ein ontologiebasiertes Produktkonfigurationssystem am Beispiel einer forstwirtschaftlichen Bohrmaschine [YDM08]. In einer „Meta-Ontologie“ werden allgemeine, disziplinunabhängige Terminologien und Beziehungen beschrieben (*meta-model layer*). Diese werden genutzt, um disziplinspezifische Konfigurations-/Variantenmodelle zu erstellen (*model layer*). Letztere können anschließend in einer konkreten Konfigurationsinstanz (*instance layer*) resultieren. Der Wissensaustausch werde durch die Wiederverwendung der Meta-Ontologie sowie bestehender Konfigurationsmodelle sichergestellt. Das Produktkonfigurationssystem wird mithilfe von OWL und SWRL umgesetzt und ist in der Lage, mögliche Kombinationen zu finden. Eine Bewertung dieser Kombinationen erfolgt nicht [YDM08].

Eine Methodik zur Generierung von OWL-Ontologien aus industriellen Klassifikationsstandards wurde von Hepp erarbeitet [Hep06]. Übergeordnetes Ziel ist es, die Aufgaben im Kontext des elektronischen Marktplatzes – wie Produktsuchen oder Ausgabenanalysen – durch die Nutzung von Semantic-Web-Technologien zu vereinfachen. Um den manuellen Aufwand bei der Erstellung der nötigen Domain Ontologies (siehe Abschnitt 2.4.3) zu reduzieren, wird daher die Überführung von Standards wie eCl@ss oder UNSPSC thematisiert. Die Methodik wurde auf den Standard eCl@ss in der Version 5.1 erfolgreich angewendet.

Zuletzt sei noch das bereits beschriebene KIEF-System erwähnt, das dem Entwickler auf Basis der topologischen Systemstruktur mögliche physikalische Effekte zur

Lösung der Aufgaben vorschlägt. Hierzu wird die sog. *physical concept ontology* verwendet, die disziplinunabhängige Beschreibungen enthält [YUT+04].

Bewertung: Ontologien haben großes Potential, sowohl was den disziplinübergreifenden Informationsaustausch, als auch, was das Auffinden von Lösungswissen angeht. Die existierenden Ansätze adressieren jedoch jeweils nur einen dieser Aspekte. Es fehlt eine Methodik, die beide Richtungen integriert und in das Vorgehen beim Entwurf mechatronischer Systeme einbindet. So kann die Abstimmung zwischen allen beteiligten Disziplinen verbessert und gleichzeitig der Lösungsraum systematisch eingeschränkt werden. Die gefundenen Lösungen müssen weiterhin bewertbar sein. Hierzu bietet sich die Nutzung von Simulationsmodellen an, die deren spezifisches Verhalten widerspiegeln. Vor diesem Hintergrund ist die Frage zu klären, wie Lösungen einerseits und Systemwissen andererseits semantisch aufzubereiten sind. Der Nutzen der semantischen Modellierung muss jedoch den Aufwand überwiegen. Aufbau und Pflege von qualitativ hochwertigen Ontologien sind vergleichsweise aufwendig, weshalb sie vor allem dort eingesetzt werden, wo Informationen von hohem Wert sind [Hor13]. Dies trifft für den Entwurf intelligenter technischer Systeme zweifelsfrei zu. Dennoch gilt es, die in [Hep07] identifizierten „Flaschenhälse“ zu berücksichtigen. Das semantische Modell muss (1) immer aktuell, (2) vom Pflegeaufwand handhabbar, (3) für den Anwender verständlich und (4) erkennbar von Nutzen sein sowie (5) gleichzeitig geistiges Eigentum respektieren.

2.6 Handlungsbedarf und Ziele der Arbeit

Zum Abschluss des Kapitels soll nun ein Fazit gezogen und der Handlungsbedarf abgeleitet werden, sodass dadurch die Ziele dieser Arbeit eingeordnet werden können.

Im Entwurfsprozess mechatronischer Systeme bestehen vielerlei Herausforderungen. Alvarez Cabrera et al. fassen diese auf Basis verschiedener Quellen zusammen [AFT+10]:

- Austausch von Entwurfsmodellen und Daten,
- kooperative Arbeit und Kommunikation zwischen den Ingenieuren,
- multidisziplinäre Modellierung,
- gleichzeitige Berücksichtigung verschiedener Disziplinen,
- frühzeitiges Testen und Verifizieren,
- beständiger, sequentieller Entwurfsprozess,
- Werkzeuge und Methoden für den disziplinübergreifenden Entwurf,

- Unterstützung beim Entwurf der Informationsverarbeitung.

Eine vollständige Automation, wie sie in [AFT+10] angestrebt wird, erscheint jedoch angesichts der beschriebenen Komplexität auf absehbare Zeit nicht erreichbar. Vielmehr soll diese Arbeit dazu beitragen, die Entwickler bei einigen dieser Aufgaben zu unterstützen. Dazu bedarf es nach der Analyse des Stands der Wissenschaft und Technik:

1) einer modellbasierten Entwurfsmethodik, welche die Integration von Lösungswissen explizit vorsieht,
2) einer effizienten Wiederverwendung von Lösungswissen und insbesondere von Dynamikmodellen,
3) der Unterstützung des Informationsaustauschs zwischen den Disziplinen während des Entwurfsprozesses.

Die Lösungsmusterspezifikation von Dumitrescu kann hierbei eine Basis bieten, da sie disziplinübergreifend angelegt ist. Um eine Bewertung ausgewählter Lösungsmuster durchführen zu können, soll sie um Dynamikmodelle erweitert werden, sodass eine möglichst frühzeitige Absicherung der Systemdynamik erfolgen kann. Wie Dynamikmodelle aufbereitet werden müssen, um sie effizient wiederverwenden zu können, wird in Kapitel 4 thematisiert. Dies ist die Grundlage, um den Entwickler beim Auffinden und beim Aufbau wiederverwendbarer Dynamikmodelle zu unterstützen. Die Ergebnisse und Beispiele werden mithilfe von Modelica dargestellt und erläutert, sind aber auch auf andere Modellierungssprachen bzw. -werkzeuge übertragbar.

Als Wissensspeicher, um zum einen das Wissen über das zu entwickelnde System (System-/Produktwissen), zum anderen das Wissen über Lösungen (Lösungswissen) zu modellieren, bieten sich Ontologien an. Mithilfe semantischer Technologien kann es dadurch anschließend im Entwurfsprozess verfügbar gemacht sowie unterschiedliche Terminologien und Strukturierungen der Disziplinen und Anbieter von Lösungswissen überwunden werden. Dabei ist darauf zu achten, dass die einzelnen Sprachen und Werkzeuge gemäß ihrer Stärken und Bestimmungen genutzt werden und der Nutzen den zu betreibenden Aufwand, zum Beispiel für die Dokumentation von Lösungswissen, überwiegt. Ziel der Arbeit ist es, ein Konzept für das reibungslose Zusammenspiel dieser Aspekte zu erarbeiten sowie es prototypisch am Beispiel umzusetzen und zu evaluieren. Hierbei sollen vornehmlich technische Kriterien berücksichtigt werden; weitergehende Potentiale in Richtung elektronischem Handel und Produktlebenszyklusmanagement (PLM) werden dabei weitestgehend ausgeklammert und nur am Rande erwähnt.

In Kapitel 3 wird dazu zunächst ein Entwurfsvorgehen vorgestellt, das die Wiederverwendung von Lösungswissen integriert (Punkt 1 des Handlungsbedarfs). Es

basiert auf dem V-Modell der VDI-Richtlinie 2206, der mechatronischen Komposition sowie auf den im Forschungsprojekt Entwurfstechnik Intelligente Mechatronik (ENTIME) miterarbeiteten Ergebnissen. Zur Erreichung einer effizienten Wiederverwendung von Lösungswissen (Punkt 2) wird darauf aufbauend die semantische Aufbereitung von Lösungselementen sowie deren Suche behandelt (Kapitel 5). Hierbei wird von einer vollständigen Beschreibung der Prinziplösung des zu entwickelnden Produkts und deren Absicherung mittels der zuvor in Kapitel 4 beschriebenen Dynamikmodelle ausgegangen. Diese stellen ebenfalls sehr wertvolles Lösungswissen dar. Auf Basis der Prinziplösung sollen Lösungselemente gefunden und integriert werden, sodass das System weiter ausgearbeitet werden kann. Zuletzt wird in Kapitel 6 ein Konzept für ein ontologie-basiertes Wissensmodell vorgestellt, welches das disziplinübergreifende Systemwissen bündelt. Hierin werden die Wechselbeziehungen der Disziplinen berücksichtigt und das übergreifend relevante Produktwissen verfügbar gemacht. Das resultierende Systemwissensmodell soll den Informationsaustausch zwischen den Disziplinen erleichtern (Punkt 3). Das Konzept wird an den kritischen Stellen prototypisch erprobt.

3 Entwurfsmethodik anhand von Anwendungsbeispielen

Das mitentwickelte Vorgehensmodell stellt eine Kombination und Erweiterung der Stärken des V-Modells der VDI-Richtlinie 2206, der mechatronischen Komposition sowie der Spezifikationstechnik CONSENS für die disziplinübergreifende Beschreibung der Prinziplösung dar (vgl. Abschnitte 2.2.1 und 2.2.2). Mithilfe von Erkenntnissen, die anhand von Anwendungsbeispielen gewonnen wurden, wird das Vorgehen des V-Modells (vgl. Bild 2-3 auf Seite 17) detailliert und ergänzt um Vorgehensschritte, welche die Integration von Lösungswissen beschreiben. Zusätzlich beinhaltet es die sog. *disziplinübergreifende Koordination*, die dafür sorgen soll, dass der Informationsaustausch sowie die Konsistenz zwischen den einzelnen Disziplinen gewährleistet bleibt. Bild 3-1 gibt eine Übersicht über die Detaillierung der einzelnen Phasen und Schritte[1]. Die Methodik[2] wird im Folgenden anhand von zwei unterschiedlichen Anwendungsbeispielen verdeutlicht. Das erste Beispiel ist die Neuentwicklung kooperierender Delta-Roboter (Abschnitt 3.1). In Abschnitt 3.2 werden anschließend wichtige Schritte bei einer Anpassungsentwicklung am Beispiel eines intelligenten Teigkneters herausgegriffen und Unterschiede erläutert. In diesem Fall können einige der dargestellten Schritte und Tätigkeiten aus der vorangegangen Entwicklung übernommen und andere übersprungen werden. Der Ablauf ist in beiden Fällen hochgradig iterativ. Das bedeutet, dass immer wieder Rücksprünge in vorangegangene Phasen nötig sind. Dies trifft insbesondere auf die jeweiligen Synthese- und Analyseschritte zu. Das sequentielle Vorgehensmodell stellt als prozessbasierte Metaisierung (siehe Abschnitt 2.1.3) dennoch eine wichtige Orientierung für die Entwickler dar (vgl. [AFT+10, PC85]).

3.1 Vorgehen bei Neuentwicklungen am Beispiel kooperierender Delta-Roboter

Der folgende Abschnitt beschreibt die entwickelte Methodik für den wissens- und modellbasierten Entwurf mechatronischer Systeme am Beispiel kooperierender

[1] In Übereinstimmung mit der Darstellung aus der VDI-Richtlinie wurde in Bild 2-3 ein flankierendes V mit der Bezeichnung „Modellbildung und -analyse“ eingezeichnet (grün hinterlegt). Vor dem Hintergrund eines durchgehend modellbasierten Entwurfsprozesses muss der Begriff „Modell“ allerdings konkretisiert werden (vgl. Abschnitt 2.1.2). Gemeint sind hier Simulationsmodelle, die gemäß der im Modell Wirkstruktur definierten Struktur aufgebaut werden und ab der Erstellung dieser zum Einsatz kommen (vgl. [VDI04]).

[2] Der Begriff Methodik wird als Gesamtheit der Methoden bei einer festgelegten Art des planmäßigen Vorgehens verstanden (vgl. [Lan15]).

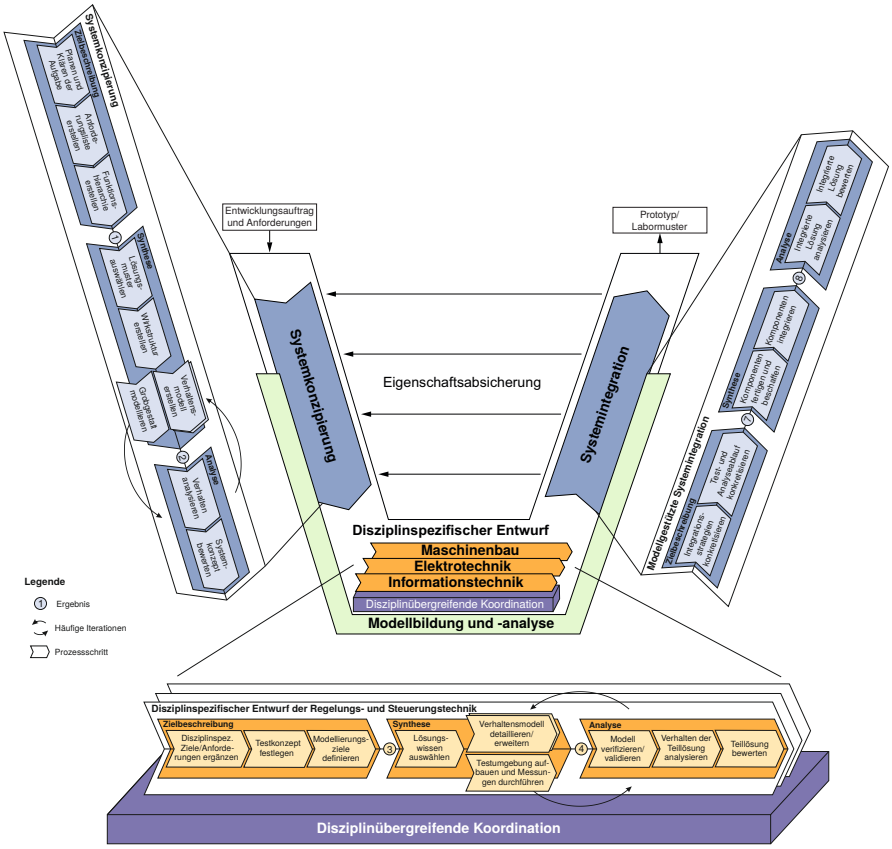


Bild 3-1: Zuordnung des Vorgehens zum V-Modell aus der VDI-Richtlinie 2206 (vgl. [Kru, Loc])

Delta-Roboter. Das Vorgehen bietet dem Entwickler einerseits eine Richtschnur beim modellbasierten Entwurf mechatronischer Systeme und bezieht andererseits die Wiederverwendung von Lösungswissen explizit mit ein. Die Ergebnisse basieren auf Arbeiten, die im Rahmen des EU-Forschungsprojekts ENTIME[3] durchgeführt wurden und in [GTS14] beschrieben sind. An dieser Stelle steht das Vorgehen im Vordergrund, tiefer gehende Details zum Anwendungsbeispiel werden daher ausgespart; sie können der angegebenen Literatur entnommen werden. Eine ausführliche Beschreibung der Entwurfsmethodik an einem industriellen Beispiel findet sich bei Kruse [Kru].

3.1.1 Entwicklungsauftrag: „Kooperierende Delta-Roboter“

Ausgangspunkte des Entwurfsprozesses sind ein konkreter Entwicklungsauftrag sowie erste Anforderungen an das zu entwickelnde System. Der Entwicklungsauftrag resultiert aus einer systematischen Produktplanung, welche u.a. Marktanalysen und Unternehmensziele berücksichtigt. Initiale Anforderungen werden von den sog. Stakeholdern[4] an das zu entwickelnde Produkt gestellt. Darüber hinaus können sie aus gesetzlichen Vorschriften resultieren oder aus früheren Entwicklungen gewonnen werden (vgl. [PBF+13]). Im konkreten Fall geht es um die Entwicklung eines Demonstrators, mit dessen Hilfe die Entwurfssystematik erarbeitet und evaluiert werden kann. Folgende Kriterien gilt es daher u. a. zu berücksichtigen (vgl. [GTS14]):

- Verwendung von am Markt verfügbaren Lösungselementen,
- dynamisch anspruchsvolle Aufgabe, die mit konventionellen Entwurfstechniken nur schwer umzusetzen ist,
- rege Kommunikation zwischen autonomen Teilsystemen, um die entwickelten Methoden der Softwaretechnik evaluieren und veranschaulichen zu können,
- Möglichkeit zur Verwendung als „Eye-Catcher“ auf Messen o. ä.

Auf Basis dieser Aufgabenstellung entstand der Auftrag, ein System aus zwei kooperierenden Delta-Robotern neu zu entwickeln, welche sich einen Ball zuspielen. Die besondere Herausforderung besteht darin, dass die Flugbahn des Balls durch keinerlei optische Systeme erfasst werden soll. Stattdessen wird sie mithilfe eines modellbasierten Beobachters geschätzt. Dieser nutzt die Aufprallkräfte, die an der Schlägerplatte gemessen werden sollen. Auf Basis der Schätzung soll dann eine

[3] Das Projekt ENTIME wurde vom Land NRW sowie der EUROPÄISCHEN UNION, Europäischer Fonds für regionale Entwicklung, „Investition in unsere Zukunft“ gefördert.

[4] *Stakeholders* sind Anspruchsgruppen aus allen internen und externen Personengruppen, die von den unternehmerischen Tätigkeiten gegenwärtig oder in Zukunft direkt oder indirekt betroffen sind [Tho15].

Vorhersage für den Aufprall auf der jeweils gegenüberliegenden Seite errechnet und an den Nachbarroboter kommuniziert werden [GTS14]. Die Informationsverarbeitung soll dezentral auf den beiden eigenständigen Robotersystemen erfolgen (AMS). Aufgrund der eingeschränkten Informationen, die zur Verfügung stehen, ist eine intensive Kommunikation zwischen den beiden Robotern erforderlich, sodass sie ein vernetztes mechatronisches System (VMS) bilden. Bild 3-2 zeigt den mechanischen Aufbau des entwickelten Demonstrators.

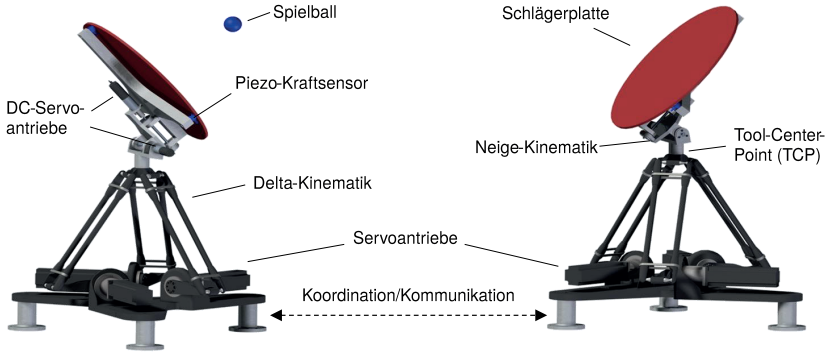


Bild 3-2: Kooperierende Delta-Roboter [DJS+ 13]

3.1.2 Disziplinübergreifende Systemkonzipierung

Die Entwicklung beginnt mit der Phase der disziplinübergreifenden Systemkonzipierung (vgl. Bild 3-3). Hier kommen Vertreter aller an der Entwicklung beteiligten Disziplinen zusammen, um erstens eine gemeinsame Vorstellung über das zu entwickelnde System zu spezifizieren und zu formalisieren, zweitens mögliche Lösungsprinzipien auszuwählen und diese drittens zu analysieren. Das Ergebnis ist die *Prinziplösung*. Sie stellt den Ausgangspunkt für die parallele, disziplinspezifische Ausarbeitung dar und definiert die disziplinspezifischen Aufgaben und Zuständigkeiten.

Zielbeschreibung

Im ersten Schritt in der Systemkonzipierungsphase – der Zielbeschreibung – geht es darum, die Aufgaben und Randbedingungen klar zu spezifizieren. Dazu werden zunächst im Zuge des *Planens und Klärens der Aufgabe* erste Anforderungs-, Funktions- und Strukturmodelle erstellt (hier u. a. Umfeldmodell und Wirkstruktur, vgl. Abschnitt 2.1.2 auf Seite 13). Als Basis fungieren die *Anwendungsszenarien*. Darin werden Betriebssituationen und das gewünschte Verhalten des

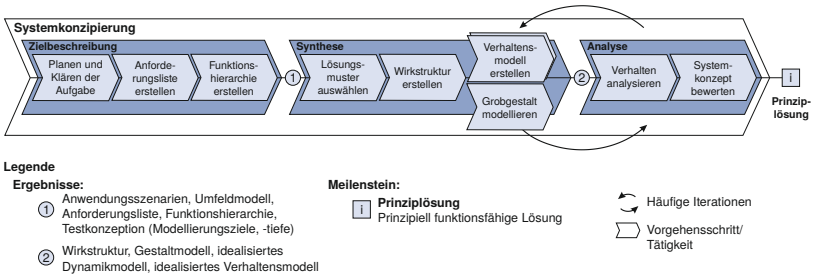


Bild 3-3: Vorgehen bei der disziplinübergreifenden Systemkonzipierung (vgl. [Kru, Loc])

Systems beschrieben. Für die kooperierenden Delta-Roboter werden acht verschiedene Anwendungsszenarien in Prosa-Steckbriefen formuliert. Beispielhaft zeigt Tabelle 3-1 das Anwendungsszenario `erfolgreiches Spielen`. Dieses beschreibt den gewünschten Ablauf des Ballspiels. Mithilfe eines Umfeldmodells werden anschließend die Wechselbeziehungen eines Roboters mit umgebenden Systeme beschrieben. Dieses Strukturmodell ist in Bild 2-10 auf Seite 29 auszugsweise dargestellt. Es beinhaltet die Wirkbeziehungen zwischen dem `Roboter 1` und seinem Umfeld, das durch die Umfeldelemente `Benutzer`, `Spielobjekt` und `Roboter 2` sowie `Energiequelle`, `Untergrund` und `Umwelt` modelliert ist. Wirkbeziehungen werden mithilfe von Energie-, Stoff- und Informationsflüssen dargestellt, die sowohl störend als auch der Funktion zuträglich sein können [GTS14].

Tabelle 3-1: Anwendungsszenario „Erfolgreiches Spielen“ [GTS14, S. 85]

AW: 5	**Erfolgreiches Spielen**	**28. Juni 2011**
Beschreibung der Situation		
Die Initialisierung der Roboter war erfolgreich und beide befinden sich im Zustand Spielen.		
Beschreibung des prinzipiellen Verhaltens		
Ein Roboter wartet auf die Soll-Parameter sowie die Aufprallprognose des anderen Roboters und empfängt diese. Der Roboter berechnet die Sollwerte für den eigenen Schlag. Das Spielobjekt wird anhand dieser Werte zum anderen Roboter geschlagen, wobei der schlagende Roboter die Aufschlagskraft während des Schlages registriert und den Wert mit dem Sollwert abgleicht. Der Soll-Ist-Abgleich hat zur Folge, dass neue Soll-Schlagparameter unter Berücksichtigung der Spielstrategie definiert werden. Diese Soll-Parameter werden wiederum wie eingangs beschrieben an den anderen Roboter versandt. Der beschriebene Ablauf wiederholt sich.		

Auf Grundlage der ersten beiden sog. Partialmodelle lassen sich systematisch weitere Anforderungen ermitteln. Sie werden in diesem Fall in einer Liste zusam-

mengefasst und nach Kategorien wie Geometrie, kontrolliertes Bewegungsverhalten, Energie, Sicherheit etc. untergliedert. Grundsätzlich besteht auch die Möglichkeit, sie zum Beispiel mithilfe von Anforderungsdiagrammen in SysML zu modellieren. Wichtig ist hier u. a. die Forderung, nur elektrische Energie zu verwenden. Diese resultiert aus dem spezifizierten Umfeld bzw. dem Anwendungsszenario, das den Einsatz auf Messen beschreibt. Dort steht in der Regel eben nur elektrische Energie bis maximal 400 V zur Verfügung. Zudem werden ein maximales Gewicht von 60 kg sowie maximale Abmaße von 1,5 x 1,5 x 1,5 m für den einzelnen Roboter festgelegt. Die Anforderungen stellen (möglichst) quantifizierbare Ziele dar, gegen welche die Eigenschaften des Systems während der gesamten Entwicklung und insbesondere in der späteren Systemintegration getestet werden. Im Laufe des Entwicklungsprozesses wird die Liste immer weiter detailliert und ergänzt.

Anschließend werden Funktionen abgeleitet, die das System erfüllen muss. Dazu wird die Hauptfunktion mittels einer *Funktionshierarchie* in Teilfunktionen untergliedert. Die Formulierung der Funktionen erfolgt unabhängig von möglichen technischen Lösungen. Dies erfordert Abstraktion und Konzentration auf die wesentlichen Probleme, die es zu lösen gilt [PBF+13]. Gleichzeitig wird das Problem solange in verschiedene Teile zerlegt, bis diese einzeln gelöst werden können (Prinzip: *teile und herrsche*). Im anschließenden Schritt der Synthese-Phase gilt es die Lösungsprinzipien bzw. Lösungsmuster auszuwählen, welche die jeweilige Funktion erfüllen.

Bild 3-4 zeigt einen Ausschnitt der Funktionshierarchie der kooperierenden Delta-Roboter. Die Hauptfunktion des Gesamtsystems lautet `Spielobjekt zuspielen`. Diese kann bis zur Funktion `Schlag ausführen` des einzelnen Delta-Roboters untergliedert werden (nicht gezeigt). Für jene Funktion sind weitere Teilfunktionen notwendig. So muss die notwendige Schlaggeschwindigkeit bestimmt, die Schlagtrajektorie berechnet und abgefahren werden. Dies erfordert eine translatorische Bewegung des Schlägers in drei Freiheitsgraden und eine Rotation der Spielplatte um zwei Achsen. Damit die translatorischen Bewegungen und die Rotationen reproduzierbar erzeugt werden können, muss die Bewegungsbahn kontrolliert werden. Hierzu werden allgemein Soll- und Sensorwerte abgerufen und mit diesen die noch nicht genauer spezifizierten Antriebe geregelt. Die Formulierung der Funktion lässt hier bewusst offen, an welcher Stelle Messwerte erfasst werden und in welchen Koordinaten die Regelung erfolgt. Hierdurch werden Vorfixierungen auf bestimmte Lösungsprinzipien verhindert und neue, innovative Ansätze ermöglicht.

Synthese und Analyse

Im Vorgehensschritt *Lösungsmuster suchen* werden Lösungsprinzipien gesucht, die eine oder mehrere Teilfunktionen erfüllen. Ziel ist es hier, pro Teilfunktion möglichst viele erfolgversprechende Lösungsprinzipien zu finden, um den Lösungsraum

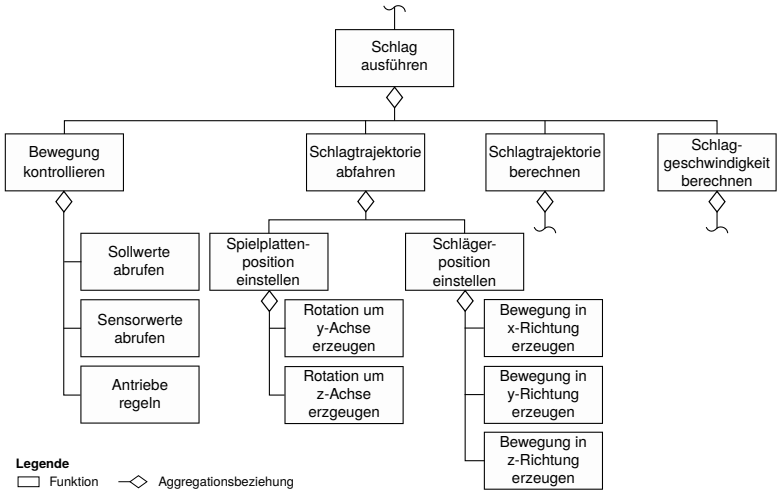


Bild 3-4: Ausschnitt der Funktionshierarchie für die kooperierenden Delta-Roboter (vgl. [GTS14, S. 86])

aufzuspannen. Sofern dieses Lösungswissen geeignet, in Form von Lösungsmustern aufbereitet wurde, kann der Entwickler hierbei durch Rechner unterstützt werden. Andernfalls muss er auf Erfahrungswissen zurückgreifen bzw. manuell recherchieren. Die effektive Wiederverwendung setzt weiterhin voraus, dass Lösungsmuster in einer Wissensbasis zur Verfügung gestellt wurden. Diese semantische Aufbereitung von Lösungsmustern ist in [GTS14] beschrieben und soll im weiteren Verlauf dieser Arbeit als gegeben vorausgesetzt werden. Anschließend werden verschiedene Lösungsvarianten erstellt, indem Kombinationen möglicher Lösungsmuster gebildet werden. Diese Kombinationsmöglichkeiten können z. B. mittels eines morphologischen Kastens [PBF+13] ermittelt werden. Für jede erfolgversprechende Lösungsvariante wird im nächsten Schritt eine *Wirkstruktur* erstellt, die alle Systemelemente und die Flussbeziehungen zwischen den Elementen enthält. Dazu wird die einem ausgewählten Lösungsmuster zugehörige Wirkstruktur (vgl. Bild 2-15 auf Seite 50) für jedes Systemelement eingesetzt, das dieses Lösungsmuster nutzt. Konnte für eine Teilfunktion kein Lösungsmuster gefunden werden, so muss hierfür eine neue Lösung entwickelt werden. In diesem Fall definiert die Wirkstruktur die Schnittstellen (vgl. Interface-Modell) des Systemelements.

Es resultieren eine oder mehrere Lösungsmöglichkeiten zur Umsetzung des Entwicklungsauftrags, deren Struktur jeweils durch die Wirkstruktur beschrieben ist. Bild 3-5 zeigt die Wirkstruktur des Systems der kooperierenden Delta-Roboter.

Jeder Delta-Roboter besteht aus einer `Delta-Kinematik`, welche mittels dreier rotatorischer Servoantriebe bewegt wird. Auf dem Tool-Center-Point der Delta-Kinematik wird ein `Schläger` mit einer Kipp- und Neigekinematik montiert, um die Schlägerplatte bei Bedarf um zwei Achsen schwenken zu können. Dies ist die Aufgabe zweier Gleichstromantriebe. Unter der Schlägerplatte befinden sich jeweils drei Piezo-Kraftsensoren, mit deren Hilfe die Sensierung der Aufprallposition erfolgt (`Ballsensierung`). Auf Basis dieser Information soll ein modellbasierter Beobachter die Ballposition kontinuierlich schätzen. Die Kenntnis der Position ist Grundvoraussetzung für die Steuerung und Regelung des Zuspiels. Nach der Spezifikation der Wirkstruktur gilt es, die Partialmodelle der Lösungsvarianten mithilfe von *Verhaltens- und Gestaltmodellen* zu vervollständigen, zu analysieren und zu bewerten. Die kohärenten Partialmodelle (vgl. Bild 2-10 auf Seite 29) bilden zusammen ein disziplinübergreifendes Systemmodell, das mithilfe des CONSENS-Profils auch durch SysML-Diagramme dargestellt werden kann [IKD+13].

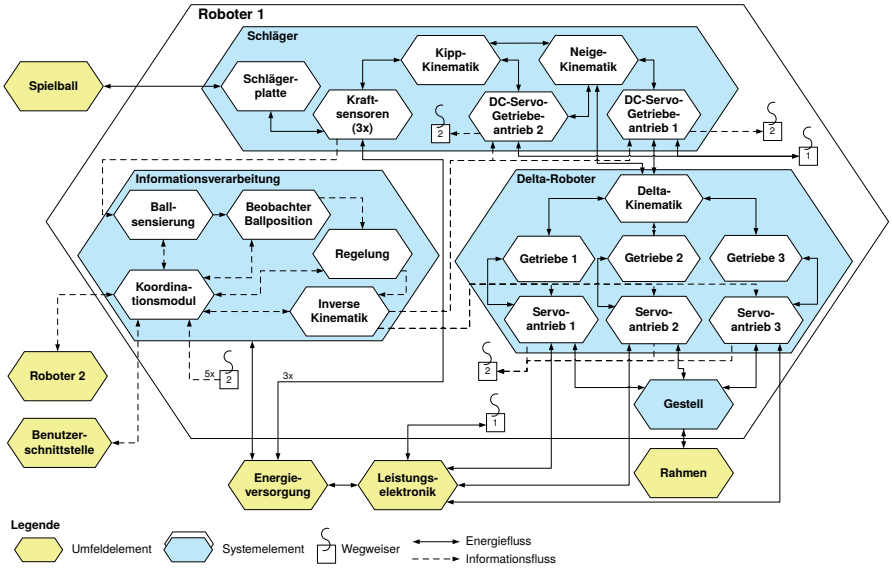


Bild 3-5: Wirkstruktur des ersten Roboters der kooperierenden Delta-Roboter

Intelligente technische Systeme zeichnen sich in vielen Fällen durch anspruchsvolle dynamische Eigenschaften aus. Hierin spiegelt sich ihr komplexer, interdisziplinärer Charakter wider, denn die Dynamik wird von mehreren Disziplinen beeinflusst. Aus diesem Grund ist es sinnvoll, die prinzipielle Funktionsfähigkeit der Systemdynamik bereits in der Konzipierung, also zu einem frühen Zeitpunkt in der Entwicklung, simulativ abzusichern. Das trifft im besonderen Maße auf das Anwendungsbeispiel

der kooperierenden Delta-Roboter zu. Ohne die Nutzung von Simulationsmodellen ist es nicht möglich, die Machbarkeit eines solchen Systems zu beurteilen. Die auftretenden Massenträgheitskräfte beim Aufprall des Balls dürfen aufgrund der notwendigen hohen Beschleunigungen nicht zu groß werden, andererseits müssen sie aber ausreichen, um die Auftreffposition mittels Kraftsensoren erfassen zu können. Es kommt folglich auf ein geeignetes Verhältnis der Massen von Schlägerplatte und Spielobjekt an.

Des Weiteren zeichnet sich die Informationsverarbeitung des Systems durch starke Kopplungen zwischen ereignisdiskretem und kontinuierlichem Verhalten aus [GTS14]. Dieses komplexe Zusammenspiel gilt es mithilfe von idealisierten Dynamikmodellen (mit geringem Detaillierungsgrad) auf der Prinzipebene abzusichern, bevor in die detaillierte Ausarbeitung gewechselt wird. Hierzu wird je Systemelement, falls vorhanden, ein entsprechend aufbereitetes Dynamikmodell (siehe Abschnitt 4.3.1) geladen und gemäß der Wirkstruktur verkoppelt (vgl. [BGK+13, BGK+12]). Dadurch reduziert sich der Modellierungsaufwand für den Anwender, weil er nun nur noch die Teilmodelle der übrigen Komponenten modellieren muss. Grundsätzlich muss hierbei nicht zwischen dem Modell der Regelstrecke und dem der Informationsverarbeitung unterschieden werden, da die Lösungsmusterbeschreibung u. a auch die Spezifikation von Mustern der Softwaretechnik erlaubt. Dennoch wird ein besonderer Aufwand in die Synthese des Verhaltensmodells der Informationsverarbeitung und der Regelung investiert werden müssen, da dieses in der Regel deutlich anwendungsspezifischer als das der Streckenkomponenten ist. Mithilfe des idealisierten Dynamikmodells des Systems können anschließend geeignete Parameterkonfigurationen ermittelt und getestet werden.

Die ermittelten Parameter konkretisieren die Komponentenanforderungen und liefern damit den Ausgangspunkt für die Suche nach Lösungselementen im Zuge der disziplinspezifischen Ausarbeitung. Synthese und Analyse erfolgen hierbei zumeist in vielen Iterationen. Die Konzipierung der Systemdynamik geschieht gemäß der Methode der mechatronischen Komposition. Die beiden Schritte *Komposition des Grundsystems* zur Auslegung des Verhaltens der Strecke und *idealisierte Komposition* zur Synthese einer ersten Informationsverarbeitung sowie zur ganzheitlichen Auslegung der Prinziplösung werden durchgeführt. Zwar können gerade die ereignisdiskreten und auf diskreten Zuständen basierenden Anteile der Informationsverarbeitung z. T. bereits vor bzw. gleichzeitig mit der Komposition des Grundsystems erfolgen, für die Konzipierung der kontinuierlichen Anteile der Steuerung und Regelung ist die Komposition des Grundsystems jedoch Voraussetzung (vgl. [Ill14]).

Im Beispiel der kooperierenden Delta-Roboter wird im Zuge der Konzipierung der Grundsystemdynamik die gewünschte Flugbahn des Balls und damit der Abstand der Roboter ausgelegt. Darüber hinaus erfolgt eine Machbarkeitsanalyse für die

Sensierung der Aufprallposition mittels Kraftsensoren. Die nötigen Massen und Steifigkeitsverhältnisse werden bestimmt. In beiden Fällen hilft ein einfaches Dynamikmodell, welches lediglich zwei ideal verstellbare Schlägerplatten und den Ball als Starrkörper sowie die Kontakte beinhaltet. Für die idealisierte Komposition wird ein Modell benötigt, das z. B. auch die idealisierte Dynamik der Antriebe und der Delta-Mechanik beinhaltet. Diese Teilmodelle stellen wiederverwendbares Lösungswissen dar, welches den ausgewählten Lösungsmustern zugeordnet werden kann (vgl. Abschnitt 4.3.1). Bild 3-6 zeigt einen Ausschnitt des erstellten idealisierten Streckenmodells der kooperierenden Delta-Roboter. Da beide Roboter exakt gleich aufgebaut sind, wird nur einer von beiden gezeigt. Für die Systemelemente `Delta-Roboter` wird jeweils das in Abschnitt 4.3.1 beschriebene, zusammengesetzte Lösungsmustermodell instanziiert und damit wiederverwendet. Dieses umfasst die Modelle der Delta-Kinematik und der Servoantriebe. Das Schlägermodell muss neu aufgebaut werden, da hierfür kein fertiges Lösungsmuster und demzufolge auch kein Lösungsmustermodell zur Verfügung steht. Das bedeutet, dass hier parallel auch die Grobgestalt konzipiert und modelliert werden muss. Die Schnittstellen liefert die Wirkstruktur. Das Modell des Spielobjekts ist über ein Kontaktmodell (Kontaktblock) mit dem des Schlägers verbunden, welcher am Tool-Center-Point (TCP) des Delta-Roboters fixiert ist (vgl. Abschnitt 4.3.2). Der Ball ist als Starrkörper modelliert, dessen Anfangswerte (Anfangsposition, -geschwindigkeit und -beschleunigung) mithilfe eines `freeMotion` Gelenks definiert sind.

Mithilfe dieses Modells der Strecke kann die Informationsverarbeitung konzipiert werden. Das bedeutet, dass nun auch ein idealisiertes Verhaltensmodell der Informationsverarbeitung aufgebaut wird. Es wird zunächst eine Reglerstruktur erarbeitet und anschließend ein Beobachter für die Ballflugbahn entworfen. Beides muss in diesem Fall bereits in der Konzipierung erfolgen, da die Machbarkeit des konzipierten Funktionsprinzips hierdurch bestimmt wird. In iterativen Analyse- und Syntheseschritten werden die Parameter des Modells grob ausgelegt, bis die Funktionsfähigkeit gegeben ist. Unter anderem existieren dadurch nun konkrete Anforderungen an die Roboter-Antriebe (Drehmoment, Drehzahl, Beschleunigung) und an die Massen der mechanischen Körper. Sofern das System positiv bewertet wird, ist die Funktionsfähigkeit des grob ausgelegten Lösungsprinzips validiert. Das Resultat wird *Prinziplösung* genannt. Sie stellt den Ausgangspunkt für die disziplinspezifische Ausarbeitung dar, indem sie darüber hinaus u. a. Zuständigkeiten und Reihenfolgen festlegt.

3.1.3 Disziplinspezifischer Entwurf

Nach dem Meilenstein *Prinziplösung* erfolgt die parallele Ausarbeitung des Systems in den einzelnen Disziplinen. Das disziplinübergreifende Systemmodell, welches

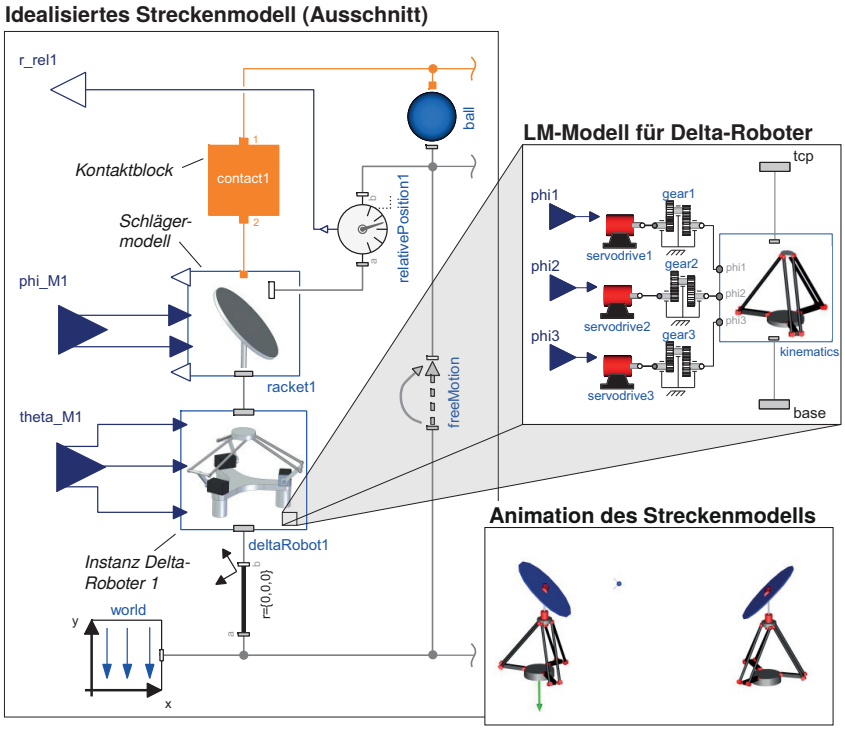


Bild 3-6: Idealisiertes Lösungsmuster-Modell der kooperierenden Delta-Roboter

sich aus den kohärenten CONSENS-Partialmodellen zusammensetzt (vgl. Bild 2-10 auf Seite 29), wird jedoch weiterhin aktuell gehalten. Die verschiedenen Disziplinen verfügen über spezifische Methoden und Vorgehensweisen, die sich etabliert haben und auch weiterhin Verwendung finden. Sie lassen sich dennoch in die übergeordneten Schritte Zielbeschreibung, Synthese und Analyse gliedern. Als Beispiel soll hier der disziplinspezifische Entwurf der Regelungs- und Steuerungstechnik beschrieben werden. Das Vorgehen ist in Bild 3-7 gezeigt.

Zielbeschreibung

Zunächst werden disziplinspezifische Anforderungen für die Ausarbeitung gesammelt und ergänzt. Diese betreffen lediglich die Zuständigkeit der beschreibenden Disziplin. Zum Beispiel werden konkrete Ziele und Anforderungen für die Sensierung/Beobachtung der Ballposition und die Bandbreite der Regelung festgelegt. In der Tätigkeit *Testkonzept festlegen* wird der Frage nachgegangen, wie und wann

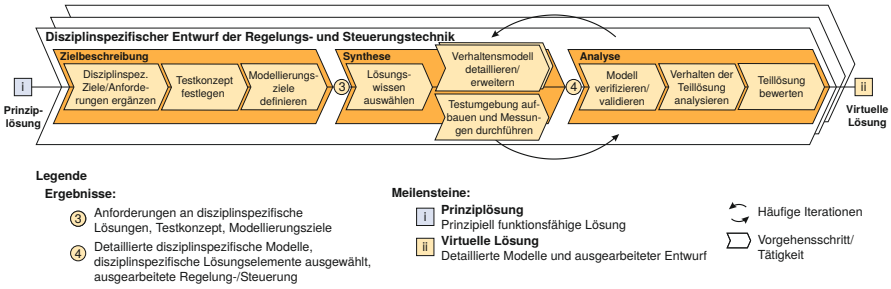


Bild 3-7: Vorgehen beim disziplinspezifischen Entwurf der Steuerungs- und Regelungstechnik (vgl. [Kru, Loc])

die entworfene Software getestet werden kann. Reicht es aus, diese ausschließlich am Modell (MiL) zu evaluieren, oder sind weitere Tests in SiL-Simulationen bzw. RCP nötig? Die Beantwortung erfordert eine Abschätzung möglicher Risiken. Grundlegend neue bzw. unsichere Funktionen sollten möglichst zuerst und am besten schnell am realen System, bzw. mit realen Komponenten abgesichert werden. Im Falle der kooperierenden Roboter konnte bei der frühzeitigen Analyse im Zuge der Konzipierung festgestellt werden, dass der Kontaktvorgang und damit die Beschaffenheit der Kontaktpartner das Verhalten stark beeinflussen. Daher wird in diesem Schritt u. a. festgelegt, dass zunächst verschiedene Bälle untersucht, d. h. in Fallversuchen vermessen werden sollen. Ziel ist es, dadurch einen geeigneten Typ auszuwählen, dessen Sensierung anschließend ausgearbeitet und möglichst schnell getestet werden kann.

Auf Basis der konkretisierten Anforderungen und des festgelegten Testkonzepts ist es im nächsten Schritt möglich, die Modellierungsziele für noch zu erstellende oder zu detaillierende Verhaltensmodelle zu definieren. Sie enthalten Zielgrößen für Modelldetaillierungsgrad und -komplexität, die abhängig sind vom erarbeiteten Testkonzept sind. Im Zusammenspiel der beiden letzten Tätigkeiten ist zu spezifizieren, wie die Modelle später identifiziert und validiert werden sollen.

Synthese und Analyse

In der Konzipierung wurden Komponentenanforderungen konkretisiert, indem anhand der idealisierten Modelle Parameter ausgelegt wurden. Zudem wurden Zuständigkeiten definiert. Im Synthese-Schritt können nun den Lösungsmustern zugehörige Lösungselemente bzw. weitere spezifische und/oder konkretere Lösungsmuster ausgewählt werden. Beispielsweise werden konkrete Lösungselemente für die Antriebe und die Delta-Kinematik durch die zuständige Disziplin ausgesucht. Gerade in der Disziplin *Regelungs- und Steuerungstechnik* existiert jedoch auch

vielerlei wertvolles disziplinspezifisches Lösungswissen, welches nicht im gleichen Maße konkret ist, sondern an die Aufgabe angepasst werden muss. In diesen Fällen soll daher weiterhin von (disziplinspezifischen) Lösungsmustern und nicht von Lösungselementen gesprochen werden. Beispiele hierfür sind Beobachter-Strukturen wie der Luenberger-Beobachter in der Regelungstechnik oder Koordinationsmuster der Softwaretechnik [Föl13, DBH+12].

Der unübersehbar große Lösungsraum bei der Suche nach Lösungswissen wird zwar durch die am idealisierten Modell gewonnenen Parameter eingeschränkt; dies kann jedoch solange nicht mit effektiver Unterstützung durch einen Rechner erfolgen, wie uneinheitliche Klassifizierungssysteme und Terminologien bestehen. Um diese Hürden zu überwinden, setzen die semantischen Technologien zur Suche nach wiederverwendbarem Lösungswissen an, die in den Abschnitten 5.1 und 5.2 detailliert beschrieben werden. Für den Fall, dass dennoch keine Lösungselemente gefunden werden (Beispiel: Schlägerkinematik), müssen die betreffenden Systemelemente neu ausgearbeitet werden. Dazu werden sie ggf. in weitere Teilsysteme untergliedert. Auch im Zuge dieser Ausarbeitung von Systemkomponenten werden Lösungsmuster ausgewählt. Die entsprechende Tätigkeit im Vorgehensmodell subsumiert all diese Fälle und wird daher allgemein mit *Lösungswissen auswählen* bezeichnet (vgl. Bild 3-7).

Zu dem wiederverwendbaren Lösungswissen zählen auch detaillierte Lösungselement-Modelle. Aufgrund einheitlicher Schnittstellen können verfügbare LE-Modelle anstelle der Lösungsmuster-Modelle eingesetzt werden (vgl. Abschnitt 4.4.1). Dies verringert den Aufwand bei der nachfolgenden Tätigkeit *Verhaltensmodell detaillieren/erweitern*. Hierunter wird wiederum sowohl die Detaillierung des Streckenmodells als auch der detaillierte Entwurf des Modells der Informationsverarbeitung im Zuge der Ausarbeitung von Steuerung und Regelung verstanden. Die einzelnen synthetisierten Bestandteil werden immer wieder iterativ analysiert und verfeinert. Auch wenn die einzelnen Teilmodelle der Lösungselemente bereits validiert worden sind, ist es dafür erforderlich, das Gesamtmodell des mechatronischen Grundsystems im Zuge der Tätigkeit *Modell verifizieren/validieren* mithilfe von Messungen oder genauen Referenzmodellen zu validieren und ggf. dessen Parameter zu identifizieren (vgl. [Kru]). Hierzu bedarf es z. B. eines Prüfmusters. Erst danach kann die Teillösung im letzten Schritt verlässlich analysiert und bewertet werden.

Bild 3-8 stellt die ausgearbeitete Reglerstruktur der kooperierenden Delta-Roboter aus Sicht kontinuierlicher und ereignisdiskreter Signale dar. Hierin werden zunächst auf Basis der Sollflughöhe und der prognostizierten Aufprallgeschwindigkeit eine Sollflugbahn und die dazu nötigen Sollwerte zur Schlägeransteuerung berechnet. Diese `Vorsteuerung` wird ergänzt durch zwei Regler für die Neigung/Verdrehung der Schlägerplatte und für die Schlägergeschwindigkeit. Sie sorgen dafür, dass die

gewünschte Höhe erreicht wird und der Ball möglichst auf die Schlägermitte trifft. Dies fußt zum einen auf einem Soll-Ist-Wert-Vergleich zwischen den aktuellen Sollgeschwindigkeiten und den beobachteten Geschwindigkeiten des vorangegangenen Schlages. Zum anderen wird die prognostizierte Aufprallposition verwendet, um die Neigung und Drehung des Schlägers zu verstellen. Sowohl die `Vorausberechnung` als auch die geschätzten Geschwindigkeiten wurden beim vorherigen Schlag in der Informationsverarbeitung des Nachbarroboters berechnet und via eines Busses kommuniziert. Der `Ballbeobachter` verwendet die gemessenen Motorwinkel sowie die Sensorsignale der Kraftsensoren unterhalb der Schlägerplatte. Diese werden im Block `Sensierung` weiterverarbeitet, sodass die benötigten Größen für den `Stoßbeobachter` bereitgestellt werden können. Der Stoßbeobachter verwendet Impulsbilanzen, um die Geschwindigkeit des Balls nach dem Aufprall zu schätzen [GTS14]. Im Bild ist darüber hinaus die Detaillierung des Streckenmodells angedeutet, welches z. B. die konkretisierten Geometrien der Disziplin Konstruktion berücksichtigt. Parallel zur Ausarbeitung der Informationsverarbeitung am Modell werden die spezifizierten *Testumgebungen aufgebaut* und die *Messungen durchgeführt*. So werden u. a. Steifigkeiten und Stoßzahlen verschiedener Spielobjekte vermessen, welche für die Berechnung in den Ballbeobachtern benötigt werden.

3.1.4 Disziplinübergreifende Koordination

Die Entwicklung intelligenter technischer Systeme, die mehr als die Summe ihrer Bestandteile darstellen (vgl. Definition auf Seite 2), kann nicht aus dem Blickwinkel einer einzelnen Fachdisziplin vorgenommen werden. Dies gilt nicht nur für die Systemkonzipierung, sondern auch und besonders für die Ausarbeitung. Die *disziplinübergreifende Koordination* dient eben jener Abstimmung und der Konsistenzsicherung zwischen den parallel arbeitenden Fachdisziplinen im Zuge der Ausarbeitung[5]. Es bedarf hierbei einer möglichst frühzeitigen und fortlaufenden Überprüfung der charakteristischen disziplinübergreifenden Hauptmerkmale und Hauptfunktionen des Systems. Dazu eignen sich insbesondere disziplinintegrierende Systemsimulationen, denn dadurch werden Eigenschaften, welche durch mehrere Disziplinen beeinflusst werden, untersucht und funktional abgesichert. Je nachdem, welche Eigenschaften analysiert werden sollen, sind unterschiedliche Disziplinen beteiligt. Zusätzlich variieren die Anforderungen und Modellierungsziele in Abhängigkeit von der zugrundeliegenden Analyseaufgabe. Im Kontext dieser Arbeit ist vor allem die disziplinübergreifende Koordination des dynamischen Verhaltens relevant. Daneben ist aber zum Beispiel eine Koordination bzgl. des Energiebe-

[5] In [Ana15] wird hierfür der Begriff fachdisziplinintegrierende Aspektspezifische-Ebene verwendet.

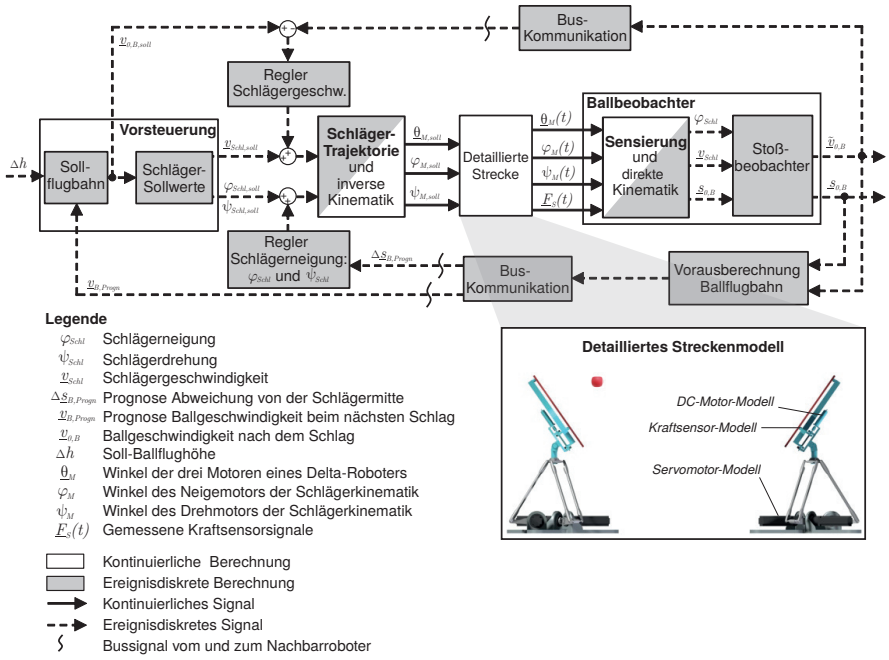


Bild 3-8: Ausgearbeitete Reglerstruktur aus Sicht kontinuierlicher und ereignisdiskreter Signale (vgl. [GTS14, S. 204])

darfs, des Bedienkomforts oder des optischen Erscheinungsbilds denkbar[6]. Auch die integrierte Simulation von Dynamik- und Verlässlichkeitsmodellen im Rahmen von Lebensdaueruntersuchungen erscheint sinnvoll (vgl. [KMS15a, KMS15b]).

Bild 3-9 zeigt das Vorgehen bzgl. der Eigenschaft Dynamik. Die entsprechenden Tätigkeiten werden in [Jus14b] zum Aufgabenbereich eines „Mechatronikers“ gezählt. Um parallel zur Ausarbeitung in den einzelnen Fachdisziplinen das Zusammenspiel zu koordinieren, werden die disziplinspezifisch ausgearbeiteten Teilsysteme vornehmlich modellbasiert integriert und mithilfe von Simulationen analysiert (MiL). Das erfordert übergreifendes Systemverständnis und Abstimmung. Der Koordinator/Mechatroniker erarbeitet daher zunächst Integrationsstrategien und -abläufe, die zum einen der Planung der modellbasierten Integration, zum anderen der Vorbereitung für die spätere Systemintegration dienen. Sie werden maßgeb-

[6] Die Identifikation der disziplinübergreifend zu koordinierenden Eigenschaften kann branchen- bzw. produktabhängig anhand der Anforderungen erfolgen. Auch die mit der Prinziplösung ausgewählte Lösungsvariante kann hierfür zusätzliche Impulse geben. Im Falle der kooperierenden Delta-Roboter sind die dynamischen Anforderungen besonders herausfordernd.

lich durch die Ausarbeitung in den einzelnen Disziplinen beeinflusst. Es werden einheitliche Schnittstellen und Strukturierungen festgelegt bzw. konkretisiert, an die beteiligten Disziplinen (z. .B. Regelungstechnik, Softwaretechnik, Elektronik, Mechanik/Maschinenbau) kommuniziert und im Systemmodell festgehalten. Im weiteren Verlauf der Zielbeschreibung werden darauf aufbauend *Test- und Analysekonzepte erarbeitet*. Die Kernfragen lauten: „Wie, in welcher Reihenfolge und mithilfe welcher Modelle können die Eigenschaften getestet werden?“ Sind Komponenten modellbasiert nur schwer analysierbar, so sollte deren Funktionsfähigkeit im Zuge der Systemintegration zusätzlich und zuerst z. B. mithilfe von HiL-Prüfständen abgesichert werden.

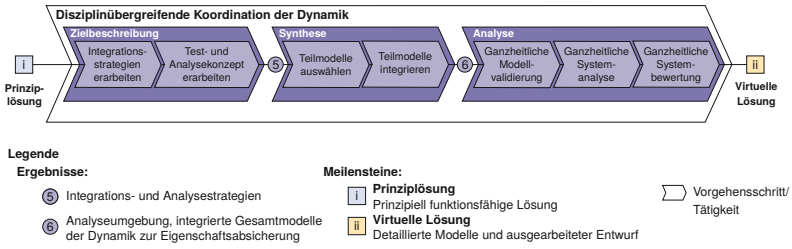


Bild 3-9: Vorgehen bei der disziplinübergreifenden Koordination bzgl. der Systemeigenschaft Dynamik (vgl. [Kru, Loc])

Im Rahmen des Synthese-Schritts werden Änderungen und neu ausgearbeitete Bestandteile, die Auswirkungen auf die Gesamtsystemdynamik haben, überwacht und gesammelt. Die beschreibenden Teilmodelle werden in der geeigneten Modellierungstiefe ausgewählt und in das Gesamtmodell integriert. Analog zum zusammengesetzten Modell des Grundsystems kann davon ausgegangen werden, dass die einzelnen Teilmodelle, welche durch die oder mithilfe der jeweiligen Fachdisziplinen erstellt werden, valide sind. Doch muss dies deshalb nicht automatisch auch für das integrierte Dynamikmodell gelten. Aus diesem Grund ist es notwendig, die Gültigkeit des Modells durch einen Vergleich mit Referenzgrößen sicherzustellen (*Ganzheitliche Modellvalidierung*). In der anschließenden *ganzheitlichen Systemanalyse* können so Konflikte und Unsicherheiten aufgedeckt werden. Es gilt zu prüfen, inwieweit die definierten Ziele gefährdet sind. So können die Teillösungen der einzelnen Disziplinen im Systemverbund hinsichtlich der Dynamik ganzheitlich analysiert und bewertet werden (ganzheitliche Komposition).

Im Beispiel der kooperierenden Delta-Roboter muss u. a. überprüft werden, ob die Verhältnisse der Massen und die Elastizität der durch den Konstrukteur ausgearbeiteten Neigemechanik geeignet sind oder den Ballabschlag negativ beeinflussen. Außerdem werden die Modelle der kontinuierlichen Regler aus der Regelungs-

technik und die der ereignisdiskreten Kommunikation aus der Softwaretechnik in einem Gesamtmodell der Informationsverarbeitung integriert und gegen ein vereinfachtes Streckenmodell getestet. Zwar führen beide Disziplinen umfangreiche Analysen durch und können Eigenschaften sogar teilweise mathematisch beweisen (Verifikation), jedoch wird vielfach erst in der ganzheitlichen Simulation offensichtlich wenn z. B. zuvor von falschen Berechnungs- oder Umschaltzeiten ausgegangen wurde. Eine kontinuierliche, integrierte Analyse des Gesamtsystems ist in der Lage, solche Fehler frühzeitig aufzudecken, bevor das System gefertigt wird. Dazu können die Modelle beispielsweise mittels FMI exportiert und in MATLAB/Simulink verkoppelt werden [GTS14, S. 222 ff.]. Ist die integrierte und ganzheitliche Überprüfung der ausgearbeiteten Teillösungen erfolgreich, so ist der Meilenstein *virtuelle Lösung* erreicht. Dieser ist Voraussetzung für die Beschaffung und Herstellung von kostenintensiver Hardware.

3.1.5 Modellgestützte Systemintegration

Nachdem die ganzheitliche modellbasierte Analyse bzgl. aller Disziplinen erfolgreich abgeschlossen ist bzw. keine weiteren Erkenntnisse mehr bringen kann, beginnt die Systemintegration. Auch hierbei können die zuvor erstellten Simulationsmodelle an vielen Stellen Unterstützung leisten. Der Ablauf der sogenannten *modellgestützten Systemintegration* als Teil der Systemintegration ist in Bild 3-10 zu sehen. Zunächst gilt es die Integrationsstrategien sowie die Test- und Analysekonzepte aus der disziplinübergreifenden Koordination aufzugreifen und mithilfe von Erkenntnissen der modellbasierten Analyse zu konkretisieren (Zielbeschreibung). Hier wird festgelegt welche Teile zunächst in Hardware aufgebaut werden und in X-in-The-Loop Simulationen getestet werden sollen. Auf Basis dieser Entscheidung können im Synthese-Schritt die erforderlichen Komponenten beschafft bzw. gefertigt und aufgebaut werden. Die integrierte (Teil-) Lösung wird anschließend analysiert und bewertet.

Während in der disziplinübergreifenden Koordination vor allem mithilfe von MiL-Simulation und im Zuge der disziplinspezifischen Ausarbeitung der Informationsverarbeitung auch mit SiL-Methoden getestet wird, kommen bei der modellgestützten Systemintegration nun HiL-Simulationen zum Einsatz. Dabei unterscheidet man Simulationen bei denen die Informationsverarbeitung im Modell simuliert und Teile des Grundsystems real aufgebaut werden (Komponenten-HiL) von dem umgekehrten Fall (Steuergeräte-HiL). In der Automatisierungstechnik spricht man an dieser Stelle von *virtueller Inbetriebnahme* und meint damit die Analyse der realen Steuerung bzw. des Steuerungscodes mithilfe eines Modells der restlichen Anlage (vgl. [SLB+13]). Durch den schrittweisen Übergang vom Modell zur Realität ist es möglich, Iterationen beim kostenintensiven Aufbau und Test

von realen Systemen zu verringern und gleichzeitig eine Gefährdung von Mensch und Maschine zu verhindern.

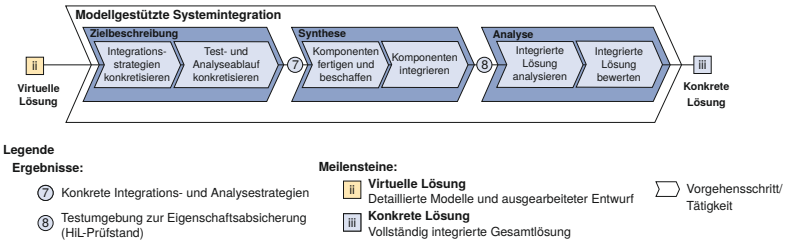


Bild 3-10: Vorgehen bei der modellgestützten Systemintegration (vgl. [Kru, Loc])

Im Beispiel der kooperierenden Delta-Roboter wird u. a. festgelegt, dass erst eine HiL-Simulation der Informationsverarbeitung auf der Zielhardware – einem Industrie-Rechner – durchgeführt werden soll. Die Stellsignale werden anschließend abgegriffen und an die realen Servoregler gegeben, sodass zunächst nur die Bewegung der Delta-Roboter unabhängig von Ballsensierung und Regelung getestet werden kann. Der Ball ist in dieser HiL-Simulation virtuell. Anschließend werden die Sensorwerte der Motorwinkel in die Informationsverarbeitung geführt. So ist eine schrittweise Realisierung möglich, welche die priorisierte Analyse kritischer Komponenten erlaubt und den mitunter schwierigen Prozess der Erstinbetriebnahme in kleinere Aufgaben unterteilt.

3.1.6 Gesamtsystemintegration (Inbetriebnahme)

Am Ende der modellgestützten Systemintegration existiert eine vollständige konkrete Lösung, die nun komplett in einem Prototyp umgesetzt werden kann. Dazu werden alle während der modellgestützten Systemintegration bereits real aufgebauten Komponenten mit den übrigen Bestandteilen in der Realität integriert und in Betrieb genommen. Im Rahmen der sogenannten *mechatronischen Applikation* erfolgt die Feinabstimmung der Parameter im realen Versuch. Beispielsweise kann es notwendig sein, Messwertfilter oder Reglerparameter anzupassen. Dazu wird zunächst das statische Verhalten und anschließend der dynamische Fall untersucht.

Das modellbasierte Testen und Entwickeln ermöglicht zwar bereits eine hohe Testabdeckung und Reproduzierbarkeit. Versuche können automatisiert, mit vergleichsweise geringem Zeitaufwand und ohne Sicherheitsrisiko durchgeführt werden. Jedoch beinhaltet jedes Modell Vereinfachungen, sodass seine Gültigkeit begrenzt ist. Daher können Untersuchungen am realen System nicht gänzlich entfallen. In der Kombination aus modellbasierten und mitunter aufwändigen realen Tests

wird eine sehr große Abdeckung der möglichen Softwarezustände erreicht. Im Zuge der Inbetriebnahme wird die Gesamtfunktionalität unter realen Bedingungen getestet und die Erfüllung der spezifizierten Anforderungen überprüft. Die zu untersuchenden Testfälle orientieren sich einerseits an den Anforderungen und andererseits an der Softwarestruktur. So können Fehler verlässlich eliminiert und die Sicherheit und Zuverlässigkeit des Systems sichergestellt werden (vgl. [Jus14b]). Insbesondere bei der Erstinbetriebnahme von Prototypen ist hier Sorgfalt geboten. Die verschiedenen Modelle im Entwurfsprozess bieten dazu zwar die Grundlage, jedoch ist darin nicht definiert, wie und in welcher Reihenfolge Funktionen in Betrieb zu setzen sind. In [WIC+15] werden daher Ansätze zur systematischen Planung und Durchführung der Inbetriebnahme mechatronischer Prototypen auf Basis des Systemmodells vorgestellt. Um größere Produktreife zu erlangen, wird der Makrozyklus des V-Modells anschließend von neuem durchlaufen (vgl. Bild 2-4 auf Seite 19).

Zum Abschluss der Entwicklungen werden vor allem die von Grund auf neu entwickelten Teile des Systems hinsichtlich möglicher Wiederverwendung untersucht. Ist dies der Fall, werden sie entsprechend der in den nachfolgenden Kapiteln beschriebenen Methodik als Lösungselemente aufbereitet und zu Lösungsmustern abstrahiert, sodass sie für zukünftige Entwicklungen zur Verfügung stehen.

3.2 Vorgehen bei Anpassungsentwicklungen am Beispiel eines intelligenten Teigkneters

Im Gegensatz zur Neuentwicklung soll das Vorgehen nun am Beispiel einer Anpassungsentwicklung erläutert werden. Im Innovationsprojekt InoTeK (**In**telligenter und **o**ptimierter **Te**ig-**K**netprozess) mit der Firma WP Kemper GmbH im Rahmen des Spitzenclusters it's OWL[7] ging es um die Anpassung und Optimierung des Teigknetprozesses in einem Spiralkneter (vgl. Bild 3-11). Spiralkneter mischen Wasser, Mehl, Hefe etc. und kneten daraus bis zu 400 kg Teig pro Charge. Hierzu werden die Zutaten in einen Bottich gegeben, der gleichsinnig zum exzentrisch gelagerten Knetwerkzeug – der Knetspirale – in Rotation versetzt wird. Die Knetspirale durchmischt die Zutaten zunächst mit langsamer Drehzahl und knetet sie anschließend mit höheren Drehzahlen zu Teig. Um den Teig in geeigneter Weise an der Spirale entlangzuführen, befindet sich zusätzlich ein Leitstab im Bottich. Dieser ist mit einem Temperatursensor (PT-100) ausgestattet, um die Temperatur

[7] Dieses Forschungs- und Entwicklungsprojekt wurde mit Mitteln des Bundesministeriums für Bildung und Forschung (BMBF) im Rahmen des Spitzenclusters „Intelligente Technische Systeme OstWestfalenLippe (it´s OWL)“ gefördert und vom Projektträger Karlsruhe (PTKA) betreut. http://www.its-owl.de/projekte/innovationsprojekte/details/intelligenter-knetprozess/

des Teigs zu überwachen. Die konventionelle Steuerung der Maschinen arbeitet rein zeitbasiert und reagiert nicht auf die variierende Qualität der Rohstoffe sowie die sich ändernden Umgebungsbedingungen. Um ein gleichbleibend gutes Ergebnis zu erzielen, ist daher das Expertenwissen geschulter Bäcker erforderlich.

Bild 3-11: Bestandteile eines Teigkneters am Beispiel des Spiralkneters „Titan“[8]

Entwicklungsauftrag: „Intelligenter Teigkneter“

Ziel ist es, eine Knetmaschine zu entwickeln, welche Informationen über den aktuellen Teigzustand aus verfügbaren Sensordaten generiert und nutzt. Weiterhin soll das Knetwerkzeug insofern verbessert werden, als das eine kostengünstigere Fertigung möglich wird. Das übrige Funktionsprinzip soll unberührt bleiben. Es handelt sich daher in diesem Fall um eine Anpassungsentwicklung, wie sie in der Praxis weitaus häufiger vorkommt als die Neuentwicklung [Rol95, S. 6]. Bei einer Anpassungsentwicklung kann auf ein Vorbild – in diesem Fall den Serienteigkneter – zurückgegriffen werden. Aus dem zugrundeliegenden Vorgehensmodell für Neuentwicklungen (vgl. Abschnitt 3.1) wird man sich folglich diejenigen Schritte herausgreifen, welche zur Erfüllung der Entwicklungsaufgabe relevant sind [PBF$^+$13]. Nachdem Voruntersuchungen sowie eine Literaturrecherche gezeigt haben, dass die Erkennung des Teigzustands prinzipiell möglich ist, steht insbesondere der disziplinspezifische Entwurf und hierbei vor allem die Ausarbeitung der intelligenten Informationsverarbeitung im Fokus.

[8] Quelle: WP Kemper GmbH

Disziplinübergreifende Systemkonzipierung

Anpassungsentwicklungen zeichnen sich i. d. R. dadurch aus, dass das grundlegende Funktionsprinzip des Systems bestehen bleibt. Insofern können große Teile der bereits existierenden Prinziplösung beibehalten werden. Dennoch gilt es, das neue Konzept disziplinübergreifend auf Basis des alten zu entwerfen – d.h. an den entscheidenden Stellen zu modifizieren – und abzusichern. Dies kann wiederum anhand von idealisierten Modellen erfolgen. Im Gegensatz zur Neuentwicklung besteht nun bereits in dieser Phase die Möglichkeit, am bestehenden System Erkenntnisse zu gewinnen, z. B. anhand von gezielten Messungen. Um heutige Knetmaschinen intelligenter zu machen, sodass sie Informationen über den aktuellen Teigzustand erfassen und auf dieser Basis den Knetprozess selbstständig führen, ist eine Anpassung der Sensorik und vor allem der Informationsverarbeitung erforderlich. Da das bewährte Funktionsprinzip – insbesondere das Prinzip der mit dem Teig in unmittelbarem Kontakt stehenden Systemelemente – möglichst unverändert bleiben soll, ergeben sich in der Konzipierung nur geringfügige Änderungen des mechanischen Grundsystems im Vergleich zur bestehenden Maschine. Die aufgebauten Partialmodelle können demzufolge weitestgehend übernommen werden. Bild 3-12 zeigt die Wirkstruktur, wobei die grün eingefärbten, zusätzlichen Systemelemente nicht Bestandteil des neuen Konzepts sind. Sie dienen der Erfassung von Größen an einem seriennahen Prüfstand im Zuge der Ausarbeitung. Die grauen Komponenten verdeutlichen wesentliche Modifikationen ggü. dem Originalsystem. Da sowohl Ergebnisse aus der Literatur (vgl. z. B. [SS06, KTT$^+$06, SCD$^+$12]) als auch zur Absicherung des Konzepts durchgeführte Voruntersuchungen gezeigt haben, dass das Teigwiderstandsmoment einerseits und die Teigtemperatur andererseits charakteristische Größen darstellen, sieht das Konzept im Unterschied zu bisherigen Maschinen die Verwendung von Servoantrieben vor. Denn diese sorgen für präzise regelbare Drehzahlen und ermöglichen neben der „serienmäßigen“ Messung der Teigtemperatur gleichzeitig auch die Erfassung des Drehmoments über den benötigten Motorstrom.

Disziplinspezifischer Entwurf

In der Ausarbeitung sind je nach Umfang der Anpassungen u. U. nicht alle Disziplinen im gleichen Maße beteiligt. In diesem Fall ist die Steuerungs- und Regelungstechnik federführend, daneben sind jedoch auch Konstruktion, Verfahrenstechnik (Bäckerhandwerk), Elektronik und Aspekte der Rheologie zu beachten. Ein disziplinspezifisches Ziel bzgl. der Informationsverarbeitung ist zum Beispiel die Erkennung des Teigoptimums mit einer maximalen Verzögerung im Bereich von 30 Sekunden. Dieses Optimum wird gekennzeichnet durch den maximalen Teigwiderstand, der unabhängig von den gewählten Drehzahlen und Teigrezepten erkannt werden muss. Inwieweit dies im realen Knetprozess überhaupt möglich ist und ob

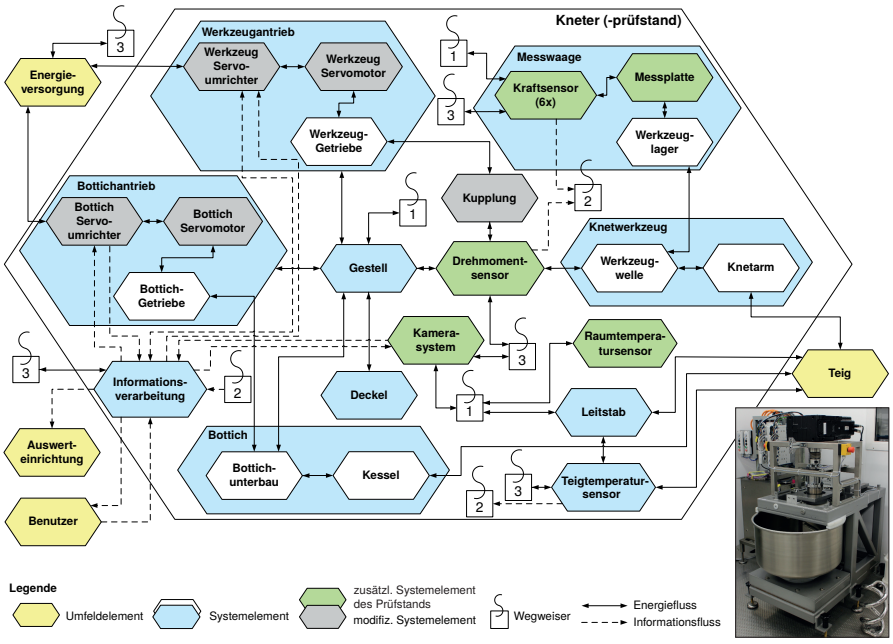


Bild 3-12: Wirkstruktur und Foto des intelligenten Kneters/Kneterprüfstands

dafür die Messung der Motorströme ausreicht, stellt die entscheidende Frage und damit das größte Risiko dar. Folglich wird ein Testkonzept erarbeitet, das einen zeitnahen Test der Software an einem Prüfmuster vorsieht (RCP). Letzteres wird darüber hinaus benötigt, um die für die Modellierung erforderlichen Messungen durchzuführen. Denn die Modellierungsziele beim Entwurf der Informationsverarbeitung sehen vor, dass vornehmlich makroskopische, phänomenologische Modelle des Grundsystems auf Basis von Messungen angestrebt werden. Diese genügen zur Beantwortung der meisten Fragestellungen und Erreichung der gesetzten Ziele und tragen darüber hinaus mit Blick auf den Modellierungsaufwand den begrenzten Ressourcen Rechnung.

Als zentraler Bestandteil des Testkonzepts wird ein seriennaher Prüfstand aufgebaut (siehe Bild 3-12, rechts). Neben dem serienmäßigen Temperatursensor, der im Leitstab angebracht ist, wird weitere Sensorik ergänzt (grün eingefärbte Systemelemente). Kräfte und Momente, die beim Knetprozess entstehen, werden mittels einer 6-Komponenten-Messwaage und eines Drehmomentsensors aufgenommen. Zusätzlich kann durch die Erfassung der Motorströme das gestellte Motormoment geschätzt und mit dem gemessen Drehmoment verglichen werden. Ein zusätzlicher

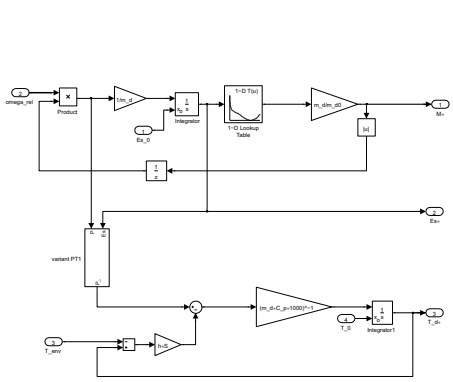
Temperatursensor misst die Umgebungstemperatur. Mithilfe dieser Testumgebung können so zunächst erforderliche Messungen durchgeführt und anschließend die kritischen Bestandteile der Informationsverarbeitung schnell getestet werden. Ungewiss ist vor allem, ob und wie gut die automatische Erkennung des Teigoptimums funktioniert. Weitere Bestandteile der Informationsverarbeitung sind eine Prädiktion, die den Endzeitpunkt sowie die dann vorherrschende Teigtemperatur vorhersagt, und die Steuerung, welche in Abhängigkeit von der Erkennung die Bewegung des Kneters koordiniert.

Koordination und Integration

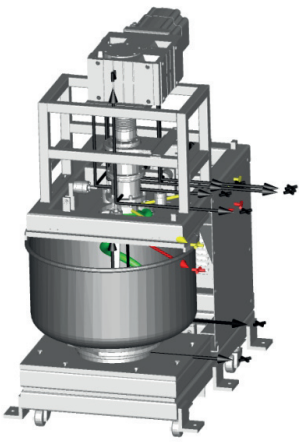
Im Zuge der Ausarbeitung des intelligenten Kneters werden viele Modelle des Systems mit unterschiedlichem Detaillierungsgrad und Modellierungszielen erstellt. Bild 3-13 zeigt einige Beispiele. Makroskopische Teigmodelle auf Basis von Messungen (Bild 3-13 (a)) dienen u. a. der Prädiktion von Teigeigenschaften in der Informationsverarbeitung, MKS-Modelle (b) werden zur Analyse und Auslegung des dynamischen Bewegungsverhaltens genutzt, CFD-Simulationen (c) geben Aufschluss über die sich einstellenden Strömungsverhältnisse, Partikelsimulationen (d) über die Vermischung. Darüber hinaus wird mittels der FEM die Festigkeit z. B. des Knetwerkzeugs untersucht. Obwohl sie sehr verschiedene Werkzeuge und Sprachen nutzen, muss die Konsistenz zwischen den Modellen stets gewährleistet sein. Dies gilt nicht nur für den Fall, dass sie direkt miteinander verkoppelt werden, sondern auch dann, wenn Ergebnisse der einen (detaillierteren) Simulation für die Modellierung in der anderen Disziplin genutzt werden sollen. Was die Temperaturen, die Teigrezeptur sowie die Knetgeschwindigkeiten und -zeiten betrifft, müssen die Modelle darüber hinaus von denselben Umgebungs- und Randbedingungen ausgehen, um effektiv ineinandergreifen zu können. Die Koordination der beteiligten Disziplinen bezieht sich jedoch nicht nur auf Modelle, sondern vielmehr auf Systemwissen allgemein, welches u. a. in Modellen gespeichert ist. Beispielsweise muss klargestellt werden, ob das verwendete Drehmoment des Motors oder das des Drehmomentsensors gemeint ist, welches um den Getriebeübersetzungsfaktor größer ist. Es ist weiterhin z.B. erforderlich, die angefertigten Messungen bzgl. ihres Zwecks und ihrer Aussagekraft zu organisieren.

Zu Beginn der modellgestützten Systemintegration werden die in der Koordination konzipierten Integrationsstrategien konkretisiert. Auch im Falle des Kneters kann schrittweise integriert werden. Nach erfolgreichem RCP-Test wird die Software auf der Zielhardware (Industriesteuerung) umgesetzt und zunächst in einer HiL-Simulation gegen das Streckenmodell getestet, welches auf einer Echtzeithardware läuft. Anschließend kann die Erkennung und Auswertung parallel zur konventionellen Ansteuerung analysiert werden, ehe zuletzt auch diese durch die neu entworfene Informationsverarbeitung übernommen wird. Beides kann zunächst

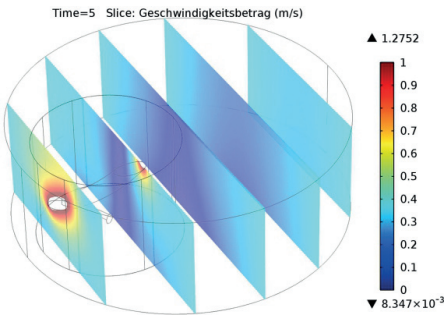
mithilfe des mechanischen Aufbaus der RCP-Testumgebung erfolgen, bevor neue Teile gefertigt und aufgebaut werden. Auch ein ggf. neu entworfenes Knetwerkzeug wird zunächst mithilfe des Prüfstands getestet, da dessen Funktionsweise und Festigkeit bei der Teigherstellung von besonderer Bedeutung ist.



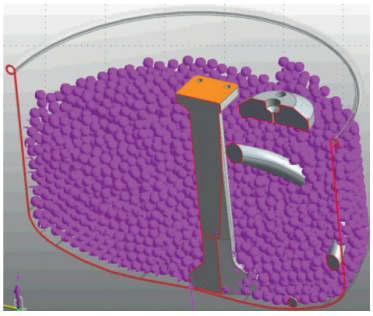
(a)



(b)



(c)



(d)

Bild 3-13: Modelle von Teig und Kneter, (a) makroskopisches Modell in MATLAB/Simulink, (b) MKS-Modell in Dymola, (c) Strömungsmodell in Comsol/Multiphysics [Tra13], (d) Partikelsimulation mit RecurDyn

4 Dynamikmodelle als Lösungswissen im Entwurf mechatronischer Systeme

Das vorangegangene Kapitel hat gezeigt, dass Dynamikmodelle einen wichtigen Erfolgsfaktor im Entwurfsprozess mechatronischer Systeme darstellen. Sie erlauben frühzeitige Analysen und verbinden die disziplinübergreifende Konzipierung mit der disziplinspezifischen Ausarbeitung. Durch (teil-)integrierte Modelle wird die Koordination und Konsistenzsicherung parallel zur fachspezifischen Ausarbeitung sowie die schrittweise Systemintegration ermöglicht. In diesem Kapitel soll es nun um die Fragestellung gehen, wie Dynamikmodelle in geeigneter Form modelliert und aufbereitet werden müssen, damit sie zielgerichtet wiederverwendet werden können und wie dabei dem Umstand Rechnung getragen werden kann, dass sich die Detaillierung im Laufe des Entwicklungsprozesses erhöht. So soll der Modellierungsaufwand reduziert und Fehler vermieden werden. Im Rahmen des ENTIME Projekts wurde dazu die Lösungsmuster-Spezifikation von Dumitrescu [Dum10] um Modellbeschreibungen erweitert[1]. Jedem Lösungsmuster werden verschieden abstrakte Modelle zugeordnet (siehe Bild 4-1). Hierdurch werden zweierlei Dinge erreicht:

1) Hat man mithilfe der Methodiken der Lösungsfindung auf Grundlage von Anwendungsszenarien, Funktionen und Anforderungen ein geeignetes Lösungsmuster identifiziert, so wird automatisch aus der Menge der vorhandenen Modelle eine Vorauswahl solcher getroffen, die zum Lösungsmuster gehören und daher zur Beschreibung des Musters geeignet sind.
2) Umgekehrt wird der Kontext und das System, für welches das Modell aufgebaut wurde, durch die Verknüpfung mit den restlichen Aspekten des Lösungsmusters sehr detailliert und umfassend beschrieben.

Dennoch können weiterhin verschiedene Modelle mit unterschiedlichen Annahmen, Abstraktionen und Grenzen zur Auswahl stehen. Der nachfolgende Abschnitt beschreibt einen Ansatz, den Detaillierungsgrad und die Komplexität von Dynamikmodellen zu quantifizieren, um sie so vergleichbar zu machen und ein Auswahlkriterium zu schaffen.

4.1 Modellierungstiefe und Modellkomplexität als quantifizierbare Auswahlkriterien

Entscheidend für den Nutzen eines jeden Modells sowie für die Erfüllung der Modellierungsziele ist die Wahl des Detaillierungsgrads (vgl. Abschnitt 2.1.1).

[1] Dies gilt im Übrigen analog auch für konkrete Lösungselemente (siehe dazu Kapitel 5).

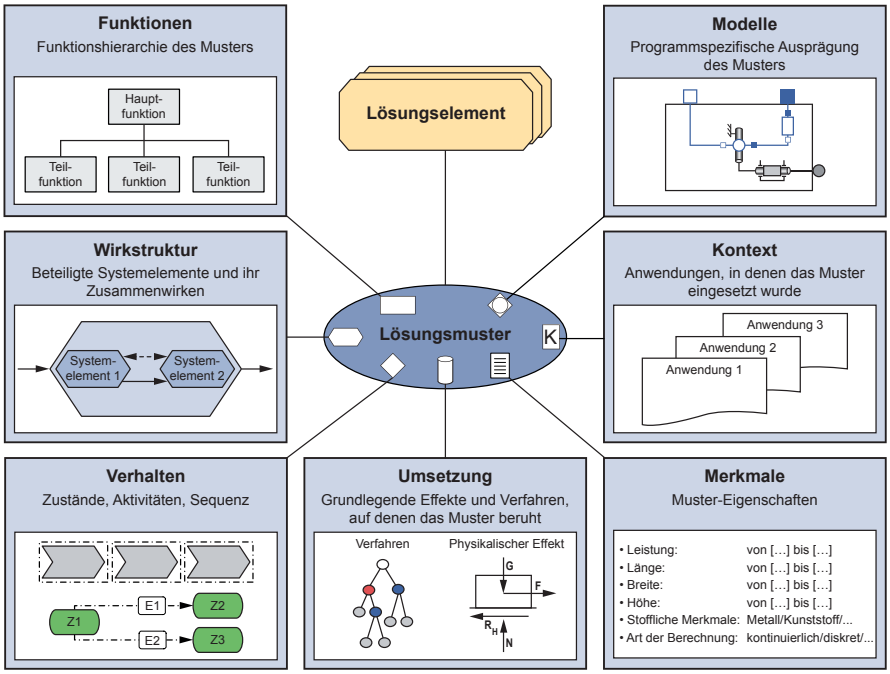


Bild 4-1: Erweiterte Spezifikation eines Lösungsmusters [GTS14, S. 125]

Dieser muss sich zum einen zur Beantwortung der Fragestellungen des jeweiligen Entwurfszeitpunkts eignen. Bei der Nutzung/Integration von bestehenden Teilmodellen kommt es zum anderen darauf an, dass die Detaillierungsgrade der Modelle zu einander passen. Letzteres wird von Brandstetter et al. als Semantik beim Modellaustausch bezeichnet [BOS00]. Es liegt also nahe, den Detaillierungsgrad eines Modells – auch *Modellierungstiefe* genannt – als ein primäres Kriterium für die Auswahl und für die Aufbereitung von Modellen als Lösungswissen zu verwenden. Die Modellierungstiefe soll im Kontext dieser Arbeit gemäß [LOT14, Loc] folgendermaßen definiert werden.

Definition 4.1 (Modellierungstiefe)**:** *Die Modellierungstiefe* Γ *eines Dynamikmodells beschreibt, wie detailliert bzw. genau physikalische Effekte und deren Wechselwirkungen abgebildet sind. Eine hohe Modellierungstiefe korrespondiert dabei mit hoher Detaillierung, sodass das Modell die Realität sehr genau (bzw. detailgetreu) wiedergeben kann. Modelle mit geringer Modellierungstiefe beinhalten hingegen viele Vereinfachungen und Annahmen.*

Nach Brooks und Tobias ist ein weiteres, wichtiges und oft verwendetes Kriterium die *Komplexität* eines Modells [BT96]. Die beiden Begriffe Modellierungstiefe und Modellkomplexität werden jedoch in der Literatur nicht immer einheitlich und zum Teil sogar synonym verwendet. Daher soll, in Anlehnung an die Abgrenzungen nach [HdF+12, BT96] u. a., die Komplexität eines Modells im Folgenden entsprechend Definition 4.2 verstanden werden (vgl. [LOT14, Loc]).

Definition 4.2 (Modellkomplexität)**:** *Die Komplexität K eines Modells ist ein strukturelles Merkmal. Eine hohe Komplexität bedeutet, dass das Modell aus einer großen Anzahl unterschiedlicher Bestandteile besteht, die viele und auch veränderliche Verbindungen untereinander besitzen. Im Gegensatz dazu haben Modelle mit geringer Komplexität nur wenige Elemente, die zudem kaum miteinander verknüpft sind.*

Beide Eigenschaften werden bei der Beschreibung eines Modells im Folgenden grundsätzlich getrennt voneinander betrachtet, hängen aber natürlich mittelbar zusammen. Denn bei Erhöhung der Modellierungstiefe wird tendenziell auch die Komplexität zunehmen. Im Hinblick auf die Modellierungsziele muss es hier das Ziel sein, den optimalen Kompromiss zwischen der geforderten Genauigkeit und der nötigen Rechenzeit zu finden.

4.1.1 Quantifizierung am Beispiel

Der Vergleich zweier Modelle anhand dieser beiden Kriterien gestaltet sich jedoch schwierig, wenn sie in Form von Prosa beschrieben sind, vor allem dann, wenn Modelle unterschiedlicher Disziplinen beteiligt sind. Ziel muss es daher sein, die Komplexität und die Modellierungstiefe von Dynamikmodellen disziplinübergreifend zu quantifizieren und so sowohl den manuellen Auswahlprozess zu unterstützen als auch die Nutzung von Informationstechnologie zu ermöglichen. Ein erster Vorschlag zur Quantifizierung beider Kriterien soll im Folgenden am Beispielmodell des Drehmomentsensors im Teigkneterprüfstand erläutert werden (siehe Bild 4-2). Im Modell werden dessen Elastizität und die jeweils anteiligen Drehträgheiten (`inertia`) sowie eine Verzögerung bei der Messwerterfassung[2] (PT1-Glied) abgebildet.

Eine relativ einfache Möglichkeit zur Quantifizierung der *Komplexität K* von Dynamikmodellen ist es, die Anzahl n der Zustandsvariablen zu zählen. Im gezeigten Beispiel entstehen je Trägheit zwei und für das Verzögerungsglied ein Zustand, also insgesamt fünf Zustände. Somit ergibt sich die Maßzahl:

$$K_{Sensor} = n_{Sensor} = 5 \tag{4-1}$$

[2] auch Gruppenlaufzeit genannt

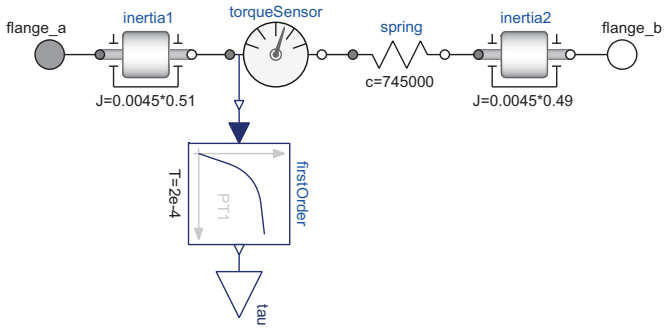


Bild 4-2: Beispielmodell eines Drehmomentsensors in Dymola/Modelica

Die Modellkomplexität kann u. a. einen Hinweis auf die zu erwartende Rechenintensität geben. Dabei werden jedoch eventuell ungünstige Formulierungen der Differentialgleichungen, bei der mehr Zustände als nötig berechnet werden müssen, nicht berücksichtigt. Gemäß der Definition 4.2 müssten sich darüber hinaus eigentlich auch die Art und Anzahl der Kopplungen zwischen den Zuständen sowie ggf. auch die Zahl der Parameter auf den Wert für die Modellkomplexität auswirken. Hierzu könnte z. B. die Besetzungsdichte der Dynamikmatrix linearer Systeme als Indikator dienen. Um die Bestimmung nicht zu aufwändig zu machen, wird an dieser Stelle jedoch darauf verzichtet. Des Weiteren beziehen sich die Ausführungen dieses Kapitels zunächst nur auf Systeme, die sich durch gewöhnliche Differentialgleichungen beschreiben lassen. Ein akademisches Beispiel, das u. a. zeigt wie sich die Systematik auf Deskriptorsysteme erweitern lässt, findet sich im Anhang (vgl. Anhang A.1).

Ansätze zur Klassifikation von Modellierungstiefen finden sich zum Beispiel bei Kufner [Kuf11] oder bei Lochbichler et al. [LSB+12]. Letztere definieren folgende, übergeordnete Kategorien (vgl. auch [SLB+13]):

1) **Ideale bzw. idealisierte Funktion:** Die geringste Modellierungstiefe ($\Gamma = 1$) heißt *idealisierte Funktion* und beschreibt den höchsten Abstraktionsgrad. Die zugehörigen Modelle bilden die ideale Funktionalität des betrachteten Systems, auf logische Art und Weise ab. Es werden nur die wichtigsten Systemgrößen modelliert und ihr zeitliches Verhalten wird vernachlässigt. Daher haben die Modelle dieser Ebene keine Zustände, ihre Komplexität ist 0. Dies trifft zum Beispiel auf die mit `torqueSensor` bezeichnete Komponente des Sensormodells in Bild 4-2 zu, da diese das zwischen den beiden Ports anliegende Drehmoment direkt und ungefiltert ausgibt (idealer Sensor).

2) **Prinzipielle Machbarkeit:** Modelle der Ebene *prinzipielle Machbarkeit* ($\Gamma = 2$) bilden zusätzlich das Zeitverhalten des Systems idealisiert ab. Diese Modelle können mit physikalisch-theoretischen Mitteln oder experimentell aufgebaut werden und dienen dazu, die grundsätzliche Funktionsfähigkeit des Systems zu untersuchen. Nebeneffekte wie zum Beispiel Reibung werden jedoch weitestgehend vernachlässigt oder sehr abstrakt modelliert. Das in Bild 4-2 dargestellte Beispiel des Drehmomentsensormodells kann dieser Kategorie zugeordnet werden, da hier u. a. auch zeitliches Verhalten in Form der Elastizität, der Drehträgheiten und der Verzögerung modelliert ist. Störende Nebeneffekte werden jedoch ausgeklammert.

3) **Systemspezifisches Verhalten:** Die Modelle der dritten Ebene ($\Gamma = 3$) werden im Normalfall größtenteils physikalisch modelliert. Nebeneffekte wie zum Beispiel Reibungseffekte oder Verlustströme werden mit einbezogen. Das spezifische Verhalten des konkreten Systems wird sehr detailgetreu wiedergegeben, sodass die Modelle dieser Ebene zur genauen Analyse des Systemverhaltens und zur detaillierten Reglerauslegung genutzt werden können. Bezogen auf das Beispiel müssten zur Erreichung dieser Ebene z. B. auch Sensorfehler (wie Linearitätsfehler, Hysterese) des konkreten Sensorsystems modelliert werden.

4) **Bauteiloptimierung:** Die vierte Ebene beschreibt Modelle mit dem geringsten Abstraktionsgrad ($\Gamma = 4$). Hier werden alle auftretenden physikalische Effekte sehr detailliert abgebildet. Die Modelle dienen der Verbesserung bzw. Optimierung eines spezifischen (Teil-)Systems im Detail mit dem Ziel, Kosten und Ressourcen einzusparen. Solche Modelle können u. a. zur Validierung von anderen Modellen geringerer Modellierungstiefe dienen. In dieser Ebene würde beispielsweise auch die berührungslose Übertragung des Messsignals des Drehmomentsensors von Rotor und Stator sowie sein magnetisches Drehzahlmesssystem abgebildet werden.

Diese Klassen der Modellierungstiefe können grob den Phasen im Entwurfsprozess zugeordnet werden, in denen sie vornehmlich zum Einsatz kommen (siehe Bild 4-3). Während in der Konzipierung Modelle der Modellierungstiefen 1 und 2 vornehmlich die Haupfunktionen des Systems wiedergeben, bilden die Klassen 3 und 4 im späteren Verlauf sukzessive weitere Nebenfunktionen und störende Effekte ab. Tendenziell steigt die erreichbare Gesamtmodellierungstiefe mit dem Fortschritt des Entwurfs, da immer detaillierteres Wissen vorhanden und nötig ist. Das bedeutet jedoch nicht, dass abhängig von den Analysezielen nicht auch einfache Modelle geringer Modellierungstiefe und -komplexität zum Einsatz kommen können. Dies ist insbesondere in der Integrations- und Koordinationsphase zu beobachten, da hier Echtzeitanforderungen beachtet werden müssen bzw. sehr große Modelle zum Einsatz kommen, bei denen lediglich die kritischen Systemelemente detailliert

werden. Andersherum spricht auch im Rahmen der Konzipierung nichts gegen die Verwendung eines geeigneten detaillierten Modells für ein Teilsystem, sofern dessen Einsatz in dieser Phase bereits feststeht.

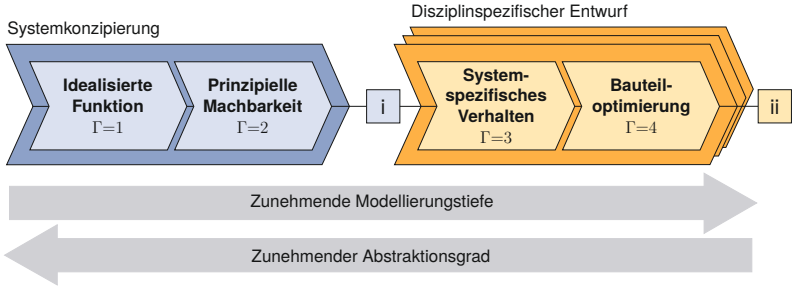


Bild 4-3: Grobe Zuordnung der maximal möglichen Gesamtmodellierungstiefen zu den Entwurfsphasen nach [LSB+12, LOT14]

Die beiden gewonnenen Größen stellen Ansätze zur quantitative Charakterisierung der Modelleigenschaften dar. Sie sollen im Folgenden genutzt werden, um die Modellauswahl zu erleichtern. Das Beispiel des Drehmomentsensors macht bereits deutlich, dass es sich bei Modellierungstiefe und Modellkomplexität um zwei unterschiedliche Kriterien handelt. Bildet man im Vergleich zu dem in Bild 4-2 gezeigten Modell der Modellierungstiefe $\Gamma = 2$ zusätzlich noch Sensorfehler mit ab, bewegt man sich in Richtung eines höheren Detailgrads. Die Anzahl der Zustände und damit die Komplexität des Modells bleibt jedoch gleich. Folglich hat eine höhere Modellierungstiefe nicht zwangsläufig eine größere Komplexität des Modells zur Folge und umgekehrt. So wird deutlich, dass zur Analyse des Sensorverhaltens im Systemkontext idealerweise ein Modell auszuwählen ist, das genau die passende Modellierungstiefe bezogen auf die Aufgabenstellung besitzt und dabei die Modellkomplexität minimiert. Um dies zu ermöglichen, müssen Dynamikmodelle, die als Lösungswissen wiederverwendet und in das Modell der Systemdynamik integriert werden sollen, geeignet aufbereitet werden. Bevor die Aufbereitung von Dynamikmodellen im folgenden Kapitel thematisiert wird, soll zunächst jedoch noch auf die Frage eingegangen werden, wie die Gesamtmodellierungstiefe eines Modells quantifiziert werden kann, wenn es aus mehreren wiederverwendeten Modellbestandteilen zusammengesetzt ist. Im Unterschied zur Komplexität des Gesamtmodells, die sich wie oben aus der Anzahl der Zustände des Gesamtmodells bestimmen lässt, stellt sich die Berechnung der Gesamtmodellierungstiefe nämlich deutlich schwieriger dar.

4.1.2 Berechnung der Modellierungstiefe zusammengesetzter Modelle

Auf der Ebene elementarer Komponenten kann die Zuordnung einer Modellierungstiefe relativ einfach erfolgen, sie geschieht anhand der obigen Definitionen. Schwieriger wird es jedoch, wenn ein Modell aus mehreren Komponenten unterschiedlicher Modellierungstiefen Γ_i besteht. Je nachdem zu welchem Zweck die Teilmodelle zu einem Gesamtmodell integriert werden, könnte man durch die Angabe des Minimums der Γ_i eine konservative Einschätzung nach unten abgeben. Dies wäre dann sinnvoll, wenn die Teile des Modells, die für die zu untersuchenden Fragestellungen benötigt werden, wenig detailliert sind. Im umgekehrten Fall könnte man das Maximum der Γ_i angeben, wenngleich die Genauigkeit eines einzelnen Modells durch ungeschickte Kombination auch verringert werden kann. Um diesem Aspekt Rechnung zu tragen und um eine allgemeingültige und realistische Bestimmung der Gesamtmodellierungstiefe Γ_{ges} zu ermöglichen, wurden in [LOT14] die Gewichtungsfaktoren γ eingeführt. Mit deren Hilfe kann Γ_{ges} für ein Modell, das aus k Bestandteilen besteht, als gewichtetes arithmetisches Mittel der einzelnen Γ_i berechnet werden (Gleichung 4-2).

$$\Gamma_{ges} = \sum_{i=1}^{k} \gamma_i \cdot \Gamma_i \tag{4-2}$$

Zur Bestimmung der γ-Faktoren scheint zunächst wiederum der Bezug zur Anzahl der modellierten Zustände naheliegend. Dies erweist sich bei genauerer Betrachtung der Anforderungen an die Gewichtungsfaktoren jedoch als ungeeignet, denn es können folgende Charakteristika spezifiziert werden:

1) Normierung: $\sum_{i=1}^{k} \gamma_i \stackrel{!}{=} 1$, mit $\gamma_i \in \mathbb{R}^{>0}$,
2) die Berechnungsvorschrift soll für alle $\Gamma = 1..4$ gültig sein,
3) die berechnete Gesamtmodellierungstiefe soll möglichst unabhängig sein von der gewählten Modularisierung bzw. der Aufteilung der Zustände in Teilmodelle und
4) für die geforderte Modellierungstiefe unnötige Zustände sollen die Gewichtung nicht unzulässig nach oben treiben.

Zur Erfüllung der letzten Forderung ist der Gedanke daher, nur diejenigen Zustände $n_{min,i}$ in die Berechnung mit einzubeziehen, die für die Erreichung der geforderten Detailtreue, welche durch die gegebenen Modellierungstiefe charakterisiert wird, mindestens erforderlich sind. Modelle unnötig hoher Ordnung werden dadurch nicht „besser gestellt“. Systemtheoretisch betrachtet und bezogen auf das Ein-/Ausgangsverhalten eines Systems bedeutet das, dass nur vollständig

steuer- und beobachtbare Zustände zur Berechnung herangezogen werden. In der Regelungstechnik kennt man hierfür den Begriff der Minimalrealisierung, deren Ordnung gleich der Anzahl der Pole der Übertragungsmatrix ist [Nou77, Sva11].

Durch diese Maßnahme bleiben die Anforderungen 1) und 2) für Teilmodelle mit $\Gamma = 1$ und dementsprechend $K = n = 0$ jedoch weiterhin verletzt. Um dem zu begegnen, wird jeweils eine Eins addiert, sodass sich der folgende Vorschlag zur Berechnung der Gewichtungen γ_i ergibt:

$$\gamma_i = \frac{1 + n_{min,i}}{\sum_{i=1}^{k}(1 + n_{min,i})} = \frac{1 + n_{min,i}}{k + \sum_{i=1}^{k} n_{min,i}} \tag{4-3}$$

Dieser stellt allerdings nur einen ersten Ansatz dar[3], denn bei genauerem Hinsehen bestehen noch immer Schwächen. So ist beispielsweise die Forderung 3) nur dann erfüllt, wenn sich die Anzahl der Module nicht ändert. Darüber hinaus bleibt unberücksichtigt, dass die Anzahl der Minimalzustände in einem zusammengesetzten Modell vielfach kleiner ist, als die Summe der $n_{min,i}$. Das passiert zum Beispiel überall dort, wo im Gesamtmodell mechanische Lagerungen eingefügt werden, durch die Bewegungsrichtungen und die zugehörigen Zustände gesperrt werden. Der Grad ihrer Detaillierung ist für das Gesamtmodell hierdurch irrelevant geworden. Zudem kann der Fall auftreten, dass derselbe Zustand in mehreren Teilmodellen beschrieben ist. Durch die Kopplung in einem Simulationswerkzeug wie Dymola und durch die dortigen Algorithmen zur Gleichungsreduktion, werden solche „doppelten“ Zustände eliminiert. Da diese in Gleichung 4-3 jedoch weiterhin gleichermaßen eingehen, sind sie in gewisser Weise überrepräsentiert. Zur genaueren Betrachtung der beschriebenen Problematiken sei an dieser Stelle jedoch auf Anhang A.1 verwiesen. Das dort behandelte akademische Beispiel befasst sich mit verschiedenen Möglichkeiten der Kopplung. Weitere Diskussionen, Erläuterungen und Herleitungen finden sich darüber hinaus in [LOT14] und vor allem in [Loc].

Zuletzt bleibt nun noch die Frage nach der praxisorientierten Bestimmung der Anzahl n_{min} der Minimalzustände zu klären. Wie bereits erwähnt, stehen diese immer im Bezug zu der durch die Modellierungsziele vorgegebenen Modellierungstiefe. Pragmatisch gesehen, kann man sich intuitiv an der Anzahl der nötigen Bewegungsfreiheitsgrade bzw. generalisierten Koordinaten orientieren. Im Falle des Drehmomentsensors (siehe Bild 4-2) ist ein Modell gefordert, welches Aussagen über die prinzipielle Machbarkeit zulässt ($\Gamma^* = 2$). Zur Erfüllung der Hauptfunktion des Sensors ist aufgrund des Funktionsprinzips eine Elastizität erforderlich. Die Drehträgheit des Sensors teilt sich auf die Eingangs- und Ausgangsseite auf.

[3] Alternativ könnte z. B. das Dominanzmaß nach Litz [Lit79] zur Ermittlung von Gewichtungen herangezogen werden.

Die beiden resultierenden Trägheitsanteile besitzen je einen Freiheitsgrad $\varphi_{1,2}$, der im Zustandsraum durch die Zustände φ_1 und $\dot{\varphi}_1$ bzw. φ_2 und $\dot{\varphi}_2$ ausgedrückt wird. Da diese bezogen auf das Ein-/Ausgangsverhalten vollständig steuer- und beobachtbar sind, ergeben sich demnach insgesamt $n_{min,\,Sensor} = 4$ Minimalzustände. Analytischer und ggf. auch durch ein Simulationswerkzeug durchführbar ist ein Ansatz, der sich mehr an der Theorie der Minimalrealisierung und an der Identifikation – in [Ise08] auch experimentelle Modellbildung genannt – orientiert. Er nutzt die für die Parameteridentifikation und Validierung ohnehin notwendigen Messungen des realen Systems (vgl. Kapitel 2.2.3). Aus den Modellierungszielen wird ein Frequenzbereich definiert, in dem das zu charakterisierende Modell den gemessenen Frequenzgang mit einer spezifizierten Genauigkeit wiedergeben muss. Für $\Gamma^* = 2$ beinhaltet dieser nur die Dynamik der für die Erfüllung der Hauptfunktionen nötigen „Hauptmassen“. Bei größeren Modellierungstiefen müssen zusätzlich weitere Frequenzbereiche und Funktionen abgebildet werden. Nun kann eine (Matrix-)Übertragungsfunktion gefunden werden, die diese Forderung als Minimalrealisierung erfüllt. Die Anzahl der Pole entspricht dann n_{min}. Bezogen auf das Beispiel des Drehmomentsensors fällt dabei im Vergleich zum Modell aus Bild 4-2 der durch das PT1-Glied (`firstOrder`) eingebrachte Pol weg, da die zugehörige Eckfrequenz mit $\omega = 5000\,\mathrm{rad/s}$ höher als die zur Analyse der prinzipiellen Machbarkeit üblicherweise geforderte Grenzfrequenz ist.

4.2 Vorgehen zur Aufbereitung und Wiederverwendung von Dynamikmodellen

Dieses Unterkapitel stellt zunächst allgemein das Vorgehen zur Aufbereitung und Wiederverwendung von Dynamikmodellen vor, bevor es in den anschließenden Abschnitten mithilfe von Beispielen verdeutlicht wird. Es wurde anhand von Modelica-Modellen entwickelt, weshalb die dargestellten Beispiele ebenfalls in Dymola/Modelica vorliegen. Das Vorgehen lässt sich aber im Grundsatz auch auf andere Sprachen und Werkzeuge übertragen. Es gliedert sich in drei übergeordnete Schritte:

1) Spezifikation von Modellierungszielen und Zielmodellierungstiefe,
2) Modellierung und Parametrierung,
3) Dokumentation und Bereitstellung.

Diese werden in den folgenden Abschnitten erläutert.

Spezifikation von Modellierungszielen und Zielmodellierungstiefe

Als ganz entscheidender Punkt hat sich die Spezifikation der Modellierungsziele herausgestellt. Gemeint ist, für welchen Zweck das Modell aufgebaut werden

soll und in welchem Kontext und Gültigkeitsbereich es genutzt wird (vgl. Abschnitt 2.1). Auf Basis der jeweiligen Problemstellung/Analyseaufgabe ergeben sich die Analyseziele. Zusätzlich zu diesen werden in den Modellierungszielen Randbedingungen berücksichtigt, die sich z. B. aufgrund der verfügbaren Software, Rechenleistung oder bzgl. der Echtzeitfähigkeit ergeben. Sie beinhalten ebenfalls Informationen über die durchzuführende Modellvalidierung und die hierfür zu verwendenden Referenzdaten. Dies gilt unabhängig von der Wiederverwendung des Modells; um eine bestimmungsgemäße Verwendung sicherzustellen, müssen die Modellierungsziele hierfür jedoch klar dokumentiert werden.

Nach der Spezifikation der Ziele werden die zu modellierenden Aspekte des Systems bestimmt und die Zielmodellierungstiefe Γ^* definiert. Diese stellt einen Kompromiss dar zwischen dem Aufwand und der nötigen Genauigkeit/Detaillierung zur Erreichung der Modellierungsziele (vgl. Bild 4-4). Durch die explizite Formulierung von Γ^* wird zum einen hoher Aufwand bei der Modellierung infolge unnötiger Detaillierung vermieden. Zum anderen wird sichergestellt, dass die Modellierungsziele (z. B. Echtzeitfähigkeit, Simulationsdauer) durch eine zu hohe Modellierungstiefe bzw. Komplexität nicht gefährdet werden (siehe Bild 4-4, insb. Verlauf 2). Bei der Definition der Zielmodellierungstiefe kann die Zuordnung aus Bild 4-3 Hilfestellung leisten, generell sollte sie zunächst eher niedrig angesetzt und falls nötig später erhöht werden. Darüber hinaus gibt Just Handlungsempfehlungen für die Wahl einer geeigneten Modellierungstiefe [Jus14a, S. 22 ff.]. Diese berücksichtigen neben der Aufgabenstellung auch das Systemwissen und die Produktreife. Ist die Produktreife hoch und die Systemkenntnis des Modellierers gering, so muss zunächst detaillierter modelliert werden, um Systemwissen aufzubauen. Soll eine allgemeine modellbasierte Optimierung des Systems erfolgen, muss die Zielmodellierungstiefe ebenfalls deutlich höher als für den Funktionsnachweis angesetzt werden. Zuletzt wird dann das geeignete Werkzeug bzw. die geeignete Modellierungssprache ausgewählt.

Modellierung und Parametrierung

Die Modellierung erfolgt im Grundsatz gemäß des Vorgehens aus Abschnitt 2.2.3 (Seite 20), welches durch Just konkretisiert wird [Jus14a]. Hier werden u. a. folgende generelle Prinzipien/Handlungsempfehlungen genannt [Jus14a, S. 24–25]:

1) klein und einfach beginnen und dann erweitern,
2) Strukturierung zur Beherrschung von Komplexität,
3) sorgfältiger Umgang mit Mess- und Simulationsdaten,
4) keine Angst vor Iterationen.

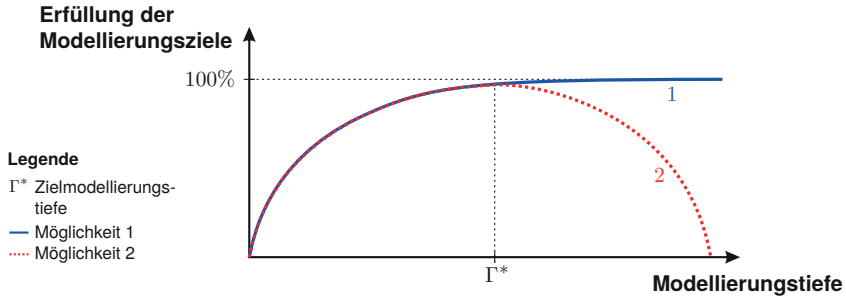


Bild 4-4: Zwei Möglichkeiten des Verlaufs des Erfüllungsgrads der Modellierungsziele bei Erhöhung der Modellierungstiefe (vgl. [LOT14, LP97])

Für die Wiederverwendung von Dynamikmodellen sollten allerdings einige Dinge ganz besonders beachtet werden. Diese Aspekte werden in Form von Modellierungsrichtlinien formuliert, welche im Folgenden auszugsweise wiedergegeben werden.

Einheitliche Schnittstellen: Im Schritt *Abgrenzung des Systems* des Vorgehens werden Schnittstellen zwischen dem System und seiner Umwelt festgelegt (vgl. Bild 2-5 auf Seite 20). Hierbei ist es sinnvoll, sich auf ggf. schon bestehende Schnittstellendefinitionen zu beziehen, bzw. neue einheitliche Schnittstellen zu definieren. Für die allermeisten Systeme wird es nützlich sein, verschiedene Modelldetaillierungsgrade vorzuhalten, da der Einsatzzweck und die damit verbundenen Genauigkeitsanforderungen variieren können. Damit die Modelle in der späteren Anwendung, z. B. im Zuge der zunehmenden Konkretisierung im Laufe des Entwurfsprozesses, ausgetauscht und die Modellierungstiefe somit variiert werden kann, ohne die Struktur des Modells zu verändern, müssen die Schnittstellen übereinstimmen. Dies gilt zum einen z. B. für den Austausch von Lösungsmustermodellen gegen Lösungselementmodelle, zum anderen analog auch für die Modelle von verschiedenen Lösungsmustern, welche dieselbe oder eine ähnliche Hauptfunktion erfüllen. Im Zuge der Lösungsfindung kann es hier erforderlich sein, verschiedene Varianten durchzuspielen und zu analysieren. Ziel muss es deshalb an dieser Stelle sein, die Modelle so aufzubauen, dass sie möglichst reibungslos gegeneinander ausgetauscht werden können. Die Auswirkung der unterschiedlichen Dynamiken kann so ohne zusätzlichen Modellierungsaufwand simuliert und verglichen werden. Die spezifizierten Richtlinien sehen daher vor, dass die Minimalschnittstellen durch das sogenannte Interface-Modell festgelegt werden, welches keinerlei weiteres Verhalten besitzt. Bei objektorientierten Modellierungssprachen wie Modelica ist deren Vererbung an die verschiedenen Modelle direkt implementierbar (siehe Bild 4-5). Im Bild wird das Interface-Modell an zwei unterschiedlich detaillierte

Modelle eines Servoantriebs vererbt. Durch die Erhöhung der Modellierungstiefe können Schnittstellen hinzukommen, z. B. ein zusätzlicher Wärmeport oder ein weiterer Eingang. Die Austauschbarkeit wird jedoch solange nicht gefährdet, wie topologieorientierte Ports verwendet werden. Werden zusätzliche Signaleingänge nötig, müssen diese deaktivierbar sein oder ein Default-Wert angegeben werden. Auf diese Art und Weise wird die Konsistenz der Schnittstellen verschiedener Modelle sichergestellt.

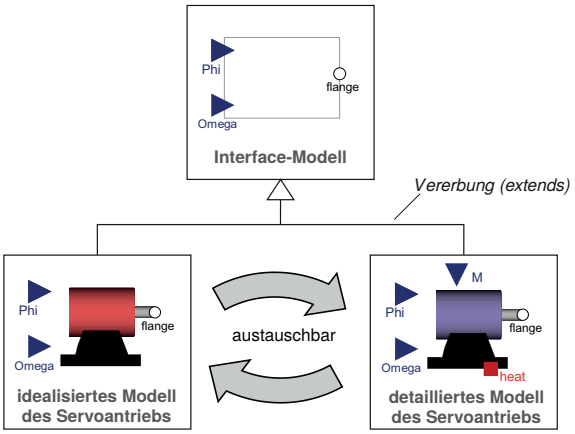


Bild 4-5: Interface-Modell zur Sicherung der Austauschbarkeit verschieden detaillierter Modelle eines Servoantriebs nach [GTS14, S. 106]

Die Schritte *Auswertung von Apriori-Kenntnissen* und *Anwendung physikalscher Gesetze* des Vorgehens zur Erstellung von Dynamikmodellen (Bild 2-5) sind potentiell fehleranfällig und aufwendig. Nach der Erstellung des physikalischen Ersatzbildes empfiehlt es sich daher, nach wiederverwendbaren Teilmodellen (Lösungswissen) für die einzelnen Komponenten Ausschau zu halten. Hierzu können semantische Technologien genutzt werden. Dem soll an dieser Stelle jedoch noch nicht vorgegriffen werden (vgl. Kapitel 5).

Übersichtliche Benennung und Gestaltung: Beim Modellaufbau hat es sich weiterhin als hilfreich erwiesen, einheitliche Konventionen für die Benennung von Variablen und Parametern einzuhalten (z. B. die in [Mod13]). So können u. a. Fehler vermieden werden, die auf unterschiedliche Schreibweisen zurückgehen. Zum Beispiel lassen sich typografische Differenzen durch Groß- und Kleinschreibung vermeiden, wenn jede Größe/Variable mit einem Kleinbuchstaben beginnt und bei einem neuen Wortstamm ein Großbuchstabe gesetzt wird (z. B. phiMotor). Diese Konvention hat gegenüber der Nutzung von Unterstrichen leichte Vorteile,

da sich so zum einen die Anzahl der Zeichen reduziert und zum anderen die Fehleranfälligkeit verringert [BDL+09]. Variablennamen sollten darüber hinaus möglichst kurz, prägnant, in einer durchgängigen Sprache (vorzugsweise in Englisch) und unabhängig vom Kontext zuzuordnen sein, gleiche oder sehr ähnliche Namen müssen vermieden werden. Auch wenn Modelle nicht angepasst sondern nur abgeschlossene Komponenten-Modelle wiederverwendet werden, sollten darüber hinaus gängige Gestaltungshinweise eingehalten werden, wie:

- Eingänge sollten links-, Ausgänge rechtsseitig positioniert werden,
- Verbindungslinien sollen sich möglichst nicht kreuzen,
- Namen oder Blöcke sollen nicht durch andere Blöcke, o. Ä. verdeckt werden,
- keine Verwendung von nicht nachvollziehbaren „Fummelfaktoren“ („magical numbers“),

Hinterlegung von Grenzen und approximierten Parametern: Bei der Wiederverwendung von Modellen ist es wichtig, sicherzustellen, dass diese nicht außerhalb ihres Gültigkeitsbereichs verwendet werden. Es lohnt sich daher, Grenzen und Randbedingungen im Modell abzubilden und den Nutzer beim Überschreiten darauf hinzuweisen. Dies trifft insbesondere auf die Parameter des Modells zu. Neben den Schnittstellen sind diese die nach außen sichtbaren Bestandteile des Modells. Für den Anwender sollte es möglich sein, das Modell mit dem zum jeweiligen Zeitpunkt im Entwurfsprozess verfügbaren Wissen zu parametrieren. Das bedeutet, dass die Parameter zum Beispiel anhand eines gängigen Datenblatts der Komponente, der spezifizierten Anforderungen und Ziele oder zumindest mithilfe von verfügbaren Referenzsignalen ermittelt werden können. Tiefgreifende, spezifische technische Kenntnisse über die verwendeten internen Bauteile der Komponente dürfen zur Parametrierung nicht vorausgesetzt werden. Parameter sollten vielmehr möglichst selbsterklärend, übersichtlich, gut dokumentiert und zentral zusammengefasst sein. Sind Parameter erforderlich, die durch den Nutzer nicht oder nur mit großem Aufwand ermittelbar sind, so müssen überschlägige Berechnungsvorschriften im Modell hinterlegt werden. Diese können entweder mithilfe bekannter Beziehungen aus der Fachliteratur, auf Basis von Erfahrungswerten aus vorherigen Projekten oder durch empirische Studien an bestehenden Systemen gewonnen werden. Die Zusammenhänge können außerdem genutzt werden, um unrealistische oder den Gültigkeitsbereich überschreitende Zustandsgrößen oder Parametrierungen zu erkennen.

Dokumentation und Bereitstellung

Entscheidend für das Auffinden und Nutzen bestehender Modelle ist eine geeignete Dokumentation. Hierbei wird zwischen Informationen unterschieden, die rein

textuell beschreibbar, sind und solchen, die derart zu annotieren sind, dass sie durch einen Rechner auswertbar werden. Außerdem kann man differenzieren zwischen Informationen, die das ganze Modell betreffen und solchen, die sich auf einzelne Parameter beziehen. Die zu dokumentierenden Eigenschaften sind in Tabelle 4-1 zusammengefasst. Einige Aspekte hieraus seien im Folgenden kurz erläutert.

Tabelle 4-1: Modelldokumentation

Bezug	**Quantifizierbar bzw. maschinell auswertbar**	**Textuell und informell**
Modell	• Name und eindeutige Identifikation • Autor/Ersteller • Datum der Veröffentlichung • Art des Modells (z. B. Zugehörigkeit zu einem Lösungsmuster/Lösungselement, Fluss) • Zugehöriges Interface-Modell • Solver • Modellierungstiefe • Komplexität • Modell validiert (Ja/Nein) • Parameterliste relevanter Parameter	• Annahmen und Vereinfachungen • Benutzungshinweise • Kontaktdaten des Erstellers (Name, Unternehmen/Organisation, Kontaktdaten)
Parameter	• Name • (Daten-) Typ • Einheit • Zahlenwert/Default-Wert • Ober- und Untergrenze • Systemrelevanz • Für eine Suche relevant (Ja/Nein) • Parameter identifiziert (Ja/Nein)	• Beschreibung • Annahmen, die der Ermittlung oder Berechnung der Default-Werte zugrunde liegen

Zur eindeutigen Identifizierung des Modells in einer Wissensbasis bedarf es neben des Namens auch einer garantiert einzigartigen Zeichenkette. Diese kann sich zum Beispiel aus dem Namen und dem Datum zusammensetzen. Sofern das Modell direkt einer Klasse, z. B. einem bestimmten Lösungsmuster oder einem Lösungselement, zugeordnet werden kann, sollte dies maschinenlesbar annotiert werden. Neben Informationen über den zu verwendenden Löser (Solver) sind die zuvor eingeführten quantitativen Auswahlkriterien Modellierungstiefe und Komplexität anzugeben. Vor der Bereitstellung eines Modells erfolgt normalerweise die Validierung. Dazu wird ein Vergleich der Simulationsergebnisse mit Messungen

am realen System oder mit bereits validierten detaillierteren Modellen durchgeführt. Jedoch kann es u. U. auch sinnvoll sein, noch nicht validierte aber aufwendige Modelle bereitzustellen. Daher ist es notwendig, dem Anwender Aufschluss darüber zu geben, ob und inwieweit das Modell validiert worden ist. Des Weiteren sollten die Parameter aufgelistet werden. Hierbei kommt es vor allem auf diejenigen Parameter an, die Systemrelevanz besitzen und/oder zur Einschränkung des Suchraums möglicher Lösungselemente verwendbar sind. Während zum Beispiel der Parameter, der bestimmt, ob das Koordinatensystem im Tool-Center-Point des Delta-Roboters (vgl. Anwendungsbeispiel kooperierende Deltaroboter) animiert wird oder nicht, auf Systemebene nur eine untergeordnete Rolle spielt, ist das Nennmoment der Antriebe für eine Reihe weiterer Komponenten des Delta-Roboters von Bedeutung. Zu diesen relevanten Größen sind jeweils Datentyp, Einheit sowie der aktuelle Wert bzw. der Default-Wert anzugeben. Weiterhin sind die Grenze der Parameter von Bedeutung, ebenso wie die Markierung, ob es sich um einen Parameter handelt, der anhand von Messungen identifiziert wurde.

Die dargestellten Informationen sind zum Teil automatisch aus einem gegebenen Modell auslesbar. An den Stellen, an denen dies nicht möglich ist, müssen sie durch den Modellersteller manuell hinzugefügt bzw. annotiert werden. Damit eine Weiterverarbeitung durch den Rechner möglich ist, bietet sich das XML-Format an. Um alle hier aufgelisteten, zu einem Modell gehörenden Metadaten (vgl. Tabelle 4-1) gesammelt in einer Datei verfügbar zu machen, wurde im Rahmen dieser Arbeit ein einheitliches XML-Schema definiert (siehe Anhang A.2). Die hiernach aufgebauten Dateien werden im Folgenden als *ModelXML* bezeichnet. Im Falle von Modelica können sie zum Beispiele mithilfe einer XSL-Transformation (siehe Anhang A.3) aus der sog. ModelicaXML Darstellung [PF03] des Modelica-Codes gewonnen werden (vgl. Abschnitt 6.3.2). Um diejenigen Informationen im Modelica-Modell kennzeichnen zu können, die nicht direkt aus dem normalen Code hervorgehen, werden spezifische Annotationen in einem Container – einem sog. `record` – eingeführt (siehe Quellcode 4-1).

Quellcode 4-1: Spezifische Annotationen

```
record ENTIMEAnnotations
parameter ModelType modeltype "type of the model";
parameter String systemElementName "name of the system element";
parameter String inheritsInterfacesFrom "interface model"
parameter Real modelingDepth "modeling depth of the model";
parameter Boolean relevant "true to specify relevant parameter";
parameter Boolean search "true to use parameter for searching";
parameter Boolean identified "true if parameter is identified";
end ENTIMEAnnotations;
```

4.3 Modelle in der Systemkonzipierung

In diesem Abschnitt wird anhand einiger Beispiele die Modellierung von idealisierten Modellen für die Konzipierungsphase gemäß des dargestellten Vorgehens näher erläutert. Dabei gibt es Modelle, die sich direkt einem Lösungsmuster zuordnen lassen (Lösungsmustermodelle). Allerdings ist auch die Wiederverwendung von anderen Modellen sinnvoll, die beispielsweise nicht zu einem Systemelement der Wirkstruktur, sondern zu einem speziellen Energiefluss zwischen diesen gehören. Ein Beispiel hierfür sind Kontaktmodelle, wie sie in Kapitel 4.3.2 vorgestellt werden. Die dargestellten Modelle nutzen die Modellierungssprache Modelica und wurden mithilfe von Dymola erstellt.

4.3.1 Lösungsmustermodelle für den idealisierten Entwurf der Systemdynamik

Für den idealisierten Entwurf der Systemdynamik kommen bereits in der Systemkonzipierung Verhaltensmodelle zum Einsatz. In dieser frühen Phase der Entwicklung stehen jedoch in der Regel noch keine genauen Informationen z. B. über das detaillierte Verhalten eines Lösungselements zur Verfügung. Um – was die Dynamik angeht – dennoch eine verlässliche Aussage über die prinzipielle Funktionsfähigkeit des Systems treffen zu können, liegt die wesentliche Herausforderung darin, die dynamischen Eigenschaften der dem Lösungsmuster zugehörigen Lösungselemente geeignet zu abstrahieren und dabei technische Randbedingungen zu berücksichtigen. Dieses Modellierungsziel vereint alle sogenannten Lösungsmustermodelle. Sie finden vornehmlich im Zuge der Konzipierung zur Analyse von Lösungsvarianten und Absicherung der Prinziplösung Verwendung. Die Zielmodellierungstiefe liegt im Bereich der *prinzipiellen Machbarkeit*, was bedeutet, dass sie größer als 1 und kleiner als 3 ist ($1 < \Gamma^* < 3$, vgl. Bild 4-3 auf Seite 90). Die Modellierung und Parametrierung von zwei Beispielen solcher Modelle sollen im Folgenden erläutert werden. Zum Abschluss des Abschnitts werden das Zusammensetzen von Lösungsmustern zu einem neuen Lösungsmuster (LM-Kaskade) sowie beispielhafte Simulationsergebnisse gezeigt. Weitere Beispiele für Lösungsmustermodelle der ENTIME-Modellbibliothek sind u. a.:

- Linear- und Spindelantriebe [Sch12a],
- Sensoren [Sch12b],
- Förderbänder [Veh14].

Idealisiertes Modell für Servoantriebe

Das erste Beispiel ist ein idealisiertes Lösungsmustermodell für elektrische, rotatorische Servoantriebe, wie sie unter anderem bei den drei Antrieben des Delta-

Roboters, aber auch zum Antrieb von Bottich und Spirale im Teigkneter Anwendung finden. Die genannten Antriebe liegen in sehr unterschiedlichen Leistungsklassen, dennoch soll auf abstrakterer Ebene dasselbe Lösungsmuster-Modell verwendbar sein.

Modellierung: Die Minimalschnittstellen eines allgemeinen Servoantriebs nach außen sind die Eingänge für den Soll-Winkel und/oder die Soll-Drehzahl sowie ein rotatorischer Mechanik-Port, der Winkel und Moment an der Motorausgangswelle beinhaltet. Diese werden als unvollständiges Modell (`partial model`) in einem Interface-Modell definiert und mittels des `extends` Befehls an das LM-Modell des elektrischen Servoantriebs vererbt. Da in der Regel zwischen Winkel- und Drehzahlregelung gewählt werden kann, werden die beiden Eingänge außerdem an Bedingungen (`phi_in, omega_in`) geknüpft, welche es ermöglichen, sie bei Bedarf zu deaktivieren (vgl. Quellcode 4-2). Werden beide Eingänge aktiviert, so entspricht das einer Winkelregelung mit Vorsteuerung der Geschwindigkeit (vgl. Zwei-Freiheitsgrade-Struktur [Föl13]). Letztere wird dabei über den Eingang `omega_in` vorgegeben und korrespondiert mit dem Soll-Winkel. Per Annotation wird der Modelltyp spezifiziert (Zeile 11). Dasselbe Interface-Modell wird ebenso an Modelle anderer Modellierungstiefe, aber auch zum Beispiel an rotatorische Servoantriebe vererbt, die hydraulische oder pneumatische Energie nutzen (vgl. Bild 4-5 auf Seite 96).

Quellcode 4-2: Interface-Modell eines Servoantriebs

```
1 partial model Servodrive "Interface Model for Servodrives"
2  parameter Boolean phi_in = true
3     "activate angle control true: active, false: inactive";
4  parameter Boolean omega_in = true
5     "activate speed control true: active, false: inactive";
6
7  Modelica.Blocks.Interfaces.RealInput omega if omega_in;
8  Modelica.Mechanics.Rotational.Interfaces.Flange_b flange;
9  Modelica.Blocks.Interfaces.RealInput phi if phi_in;
10
11   annotation (ENTIME(modeltype=InterfaceModel),...);
12 end Servodrive;
```

Das idealisierte Simulationsmodell des Lösungsmusters *Servoantrieb* ist in Bild 4-6 dargestellt. Der elektrische Teil des Motors ist lediglich durch die konstante elektro-mechanische Kopplung mit der Motorkonstanten K_M abgebildet. Der mechanische Teil wird durch die Rotorträgheit modelliert. Die Reibung wird als viskose Lagerreibung angenommen, bei der das Reibmoment linear mit der Drehgeschwindigkeit steigt. Die Regelung erfolgt bei heutigen Servoantrieben im Normalfall durch eine Kaskadenregelung. Der innerste Stromregelkreis wird in diesem idealisierten Modell durch eine Verzögerung erster Ordnung approximiert.

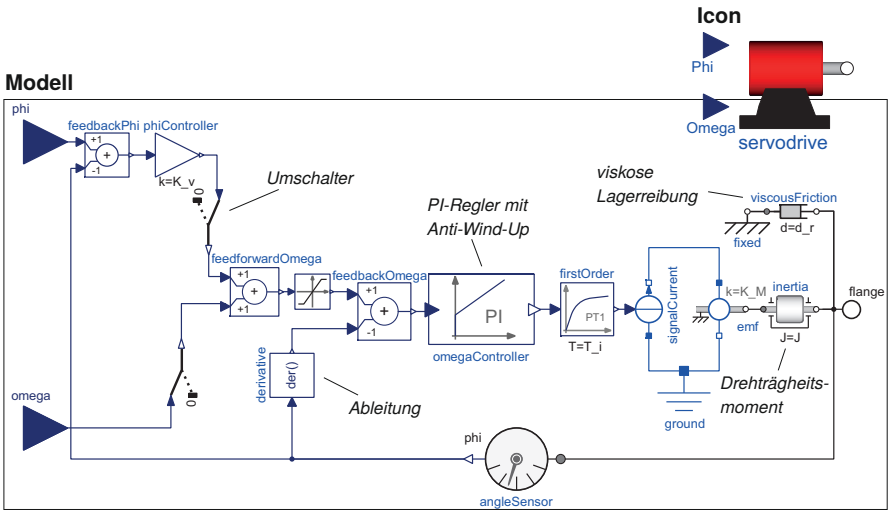


Bild 4-6: Idealisiertes Lösungsmustermodell eines Servoantriebs

Hierzu wird ein Verzögerungsglied erster Ordnung (`firstOrder`) eingesetzt. Für den Drehzahlregelkreis wird ein PI-Regler mit Anti-Wind-Up-Funktion verwendet. Für den Winkelregelkreis bedarf es zur Erreichung stationärer Genauigkeit keines zusätzlichen I-Anteils, weshalb hier ein P-Regler zum Einsatz kommt. In beiden Reglern wird eine Sollwertbegrenzung vorgesehen. Dieser kommt große Bedeutung zu, da sie verhindert, dass im Modell der technisch realisierbare Arbeitsbereich verlassen wird. Die Sensorik wird als ideal angenommen, d. h. dass zum Beispiel keine Verzögerungen bei der Ist-Wert-Erfassung betrachtet werden. Diese stellen ungewünschte Effekte dar, die in der gewünschten Zielmodellierungstiefe von 2 nicht berücksichtigt werden. Das LM-Modell des Servoantriebs besitzt 4 Zustände bei 2 Minimalzuständen[4], sodass sich folgende Werte für Komplexität und Modellierungstiefe ergeben:

[4]Zur Approximation des Frequenzgangs des geschlossenen Lageregelkreises eines vorhandenen Servoantriebs mit der Modellierungstiefe $\Gamma^* = 2$ genügt im einfachsten Fall ein Verzögerungsglied 2. Ordnung mit 2 Zuständen (vgl. [ZW06, S. 286]). Hierbei kann jedoch der direkte Zusammenhang zu den Parametern und Leistungsdaten nicht hergestellt werden, sodass diese auch nicht zur Einschränkung des Lösungsraums verwendet werden können. Auch die Grenzen des Antriebs lassen sich so nicht abbilden. Darüber hinaus sind Messungen und experimentelle Modellbildung in der Konzipierungsphase zumeist nicht möglich, da das Lösungskonzept zu diesem Zeitpunkt noch nicht real aufgebaut ist. Im gezeigten Modell entstehen folglich mehr als 2 Zustände durch: Drehwinkel, Drehgeschwindigkeit, Strom (`firstOrder`) und I-Anteil im Geschwindigkeitsregler.

$$K_{Antrieb} = n_{Antrieb} = 4$$
$$\Gamma_{Antrieb} = 2$$

Parametrierung: Lösungsmustermodelle sollen bereits mit wenigen, einfach zu setzenden Parametern simulierbar sein und dennoch realistische Ergebnisse liefern. Es soll sich um Größen handeln, die der Entwickler in der Konzipierungsphase und auf Grundlage der Systemspezifikation zumindest abschätzen kann. Zur Simulation des LM-Modells *Servoantrieb* müssen mindestens folgende Parameter angegeben werden, die sich aus den Komponentenanforderungen ermitteln lassen:

- Nenndrehmoment M_N,
- Maximalmoment M_{Max},
- Drehmomentkonstante K_M,
- Nenndrehzahl ω_N,
- Versorgungsspannung U_{Net},
- (Äquivalentes) Lastmoment J_{Load}.

Auf dieser Basis erfolgen überschlägige Berechnungen für die Default-Werte der übrigen Parameter. Hierzu werden Berechnungsvorschriften im Modell hinterlegt. Zur Ermittlung der Zeitkonstante des Stromregelkreises T_i wurden Parameterstudien durchgeführt, die zu einer quadratischen Regressionskurve in Abhängigkeit von M_{Max} geführt haben [Sch12a]. Weiterhin erfolgt eine automatische Berechnung der Reglerparameter des Geschwindigkeitsreglers (PI-Regler) mithilfe des in der Antriebstechnik gängigen symmetrischen Optimums [Föl13]. Hierzu wird die gesamte anzutreibende Drehträgheit benötigt, die sich aus dem (äquivalenten) Trägheitsmoment der Last und dem Rotorträgheitsmoment zusammensetzt. Während der Anwender die Last J_{Load} in der Regel relativ gut kennt, ist das Rotorträgheitsmoment J_R ein vielfach unbekannter Parameter. Er kann jedoch auf Grundlage einer logarithmisch, in Abhängigkeit der Bemessungsleistung aufgetragenen Kennlinie für gängige Servomotoren aus [Kie07, S. 245] abgeschätzt werden. Dazu werden die Werte für Nenndrehmoment, -drehzahl und Maximalmoment genutzt. Der viskose Reibungsbeiwert in der Lagerung der Motorwelle kann auf ähnliche Art und Weise mithilfe von überschlägigen Berechnungsvorschriften [Sch15] sowie Default-Werten für die kinematische Viskosität des Schmierstoffs, des Lagerbeiwerts und des Wellendurchmessers ermittelt werden (vgl. [Sch12a]). Die maximale Drehzahl wird [ZW06] folgend gemäß $\omega_{Max} = U_Z/K_S \approx 1{,}35 \cdot U_{Net}/K_M$ berechnet.

Idealisiertes Modell einer Delta-Kinematik

Ein weiteres Lösungsmustermodell, das bei der Dynamikmodellierung der kooperierenden Delta-Roboter zum Einsatz kommt, ist das Modell einer Delta-Kinematik (vgl. Bild 3-2 und Bild 3-5). Es wird jeweils für das entsprechende Systemelement der beiden identischen Roboter eingesetzt. Die Delta-Kinematik wurde als spezielle Parallelkinematik erstmals von Clavel vorgeschlagen [Cla91]. Sie stellt eine relativ einfache Parallelkinematik dar, die sich aus drei identischen Armen zusammensetzt. Diese sind in einem Winkel von jeweils 120° zueinander an einer Grundplatte angebracht und stellen die Verbindung zum Endeffektor bzw. Tool-Center-Point (TCP) her. Durch das Bilden von Parallelogrammen wird eine Rotation des Endeffektors verhindert und es entstehen drei translatorische Freiheitsgrade. Diese können durch die Antriebe in den drei rotatorischen Gelenken der Grundplatte beeinflusst werden. Parallelkinematiken im Allgemeinen und die Delta-Kinematiken im Speziellen zeichnen sich, aufgrund ihrer geschlossenen kinematischen Ketten, durch hohe Steifigkeit, Positioniergenauigkeit und eine geringe bewegte Masse aus. Letzteres ermöglicht große Beschleunigungen des Endeffektors. Sie werden daher besonders zur Erfüllung sogenannter *Pick-and-Place*-Aufgaben mit kleinen Taktzeiten eingesetzt, bei denen Bauteile aufgenommen und möglichst schnell und genau an anderer Stelle wieder abgesetzt werden müssen. Ein Nachteil hierbei ist jedoch ihr vergleichsweise kleiner Arbeitsraum. Bild 4-7 stellt den prinzipiellen Aufbau einer Delta-Kinematik nach Clavel dar.

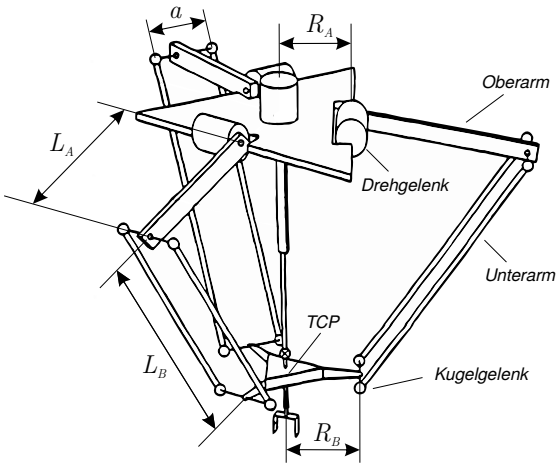


Bild 4-7: Prinzip einer Delta-Kinematik nach Clavel *[Cla91, S. 11] (vom Autor verändert)*

Modellierung: Bild 4-8 zeigt das Teilmodell eines Arms im MKS „Delta-Kinematik“. Dieses beginnt auf der linken Seite mit dem angetriebenen Drehgelenk, das über die Schnittstelle `frame_a` mit der Grundplatte sowie über den rotatorischen Port `flange` mit dem Antrieb verbunden werden kann. Der Oberarm wird über den Starrkörper `upperArm` abgebildet, an dessen Ende die Aufspreizung (`fixedTranslation`) in die beiden Unterarme erfolgt. Die zwei Stäbe (`lowerArm`) stellen die Verbindung zu der Endeffektor-Plattform her. In realen Umsetzungen geschieht dies durch zwei Kugelgelenke an den Enden. Eines davon wird im Modell allerdings durch ein Kardangelenk ersetzt, um die Rotation des Stabes um seine eigene Achse zu unterbinden. Dies stellt jedoch keine Einschränkung der Bewegungsfreiheitsgrade der Delta-Kinematik dar. Das geschlossene Parallelogramm wird über die Anbindung an die jeweiligen Koppelpunkte der Endeffektorplattform gebildet. Das Modell beinhaltet somit kinematische und dynamische Aspekte von Delta-Kinematiken, jedoch keine Nebeneffekte wie Reibung. Ggf. kann aber eine gewisse Elastizität der Struktur durch ein rotatorisches Feder-Dämpfer-Element im Antriebsstrang eingebracht werden. Die Modellierung erfolgte demzufolge mit der Modellierungstiefe *prinzipielle Machbarkeit*, was dem zuvor definierten Ziel entspricht. Pro Freiheitsgrad der Kinematik ergeben sich zwei Zustände[5], sodass gilt:

$$K_{Kinematik} = n_{Kinematik} = n_{min,\,Kinematik} = 6$$
$$\Gamma_{Kinematik} = 2$$

Dies entspricht der Darstellung in generalisierten Koordinaten (Minimalrealisierung). Um die direkte Ansteuerung in den Endeffektorkoordinaten $\underline{r}_{TCP}$ zu ermöglichen, wird zusätzlich auch das Verhaltensmodell der *inversen Kinematik* dem Lösungsmuster zugeordnet (Gleichung 4-4). Diese berechnet aus dem gegeben Ortsvektor $\underline{r}_{TCP}$ der TCP-Position die nötigen Gelenkwinkel $\theta_i, i = 1..3$ (siehe hierzu u. a. [Pra12]).

$$\underline{q}(t) = \begin{Bmatrix} \theta_1(t) \\ \theta_2(t) \\ \theta_3(t) \end{Bmatrix} = f(\underline{r}_{TCP}, t)\,, \text{ mit } \underline{r}_{TCP}(t) = \begin{Bmatrix} x_{TCP}(t) \\ y_{TCP}(t) \\ z_{TCP}(t) \end{Bmatrix} \tag{4-4}$$

Parametrierung: Der Entwickler kennt in der Regel den notwendigen Arbeitsraum des Roboters bzw. der Kinematik, ausgehend von den Anforderungen an das zu entwickelnde mechatronische System. Um die zur Erreichung dieses Arbeitsraums nötigen optimalen Armlängen angeben zu können, ist jedoch detailliertes

[5] Je nachdem wie die dynamische Wahl der Zustandsgrößen erfolgt, sind das z. B. die Gelenkwinkel θ_i und die zugehörigen Geschwindigkeiten $\dot{\theta}_i$

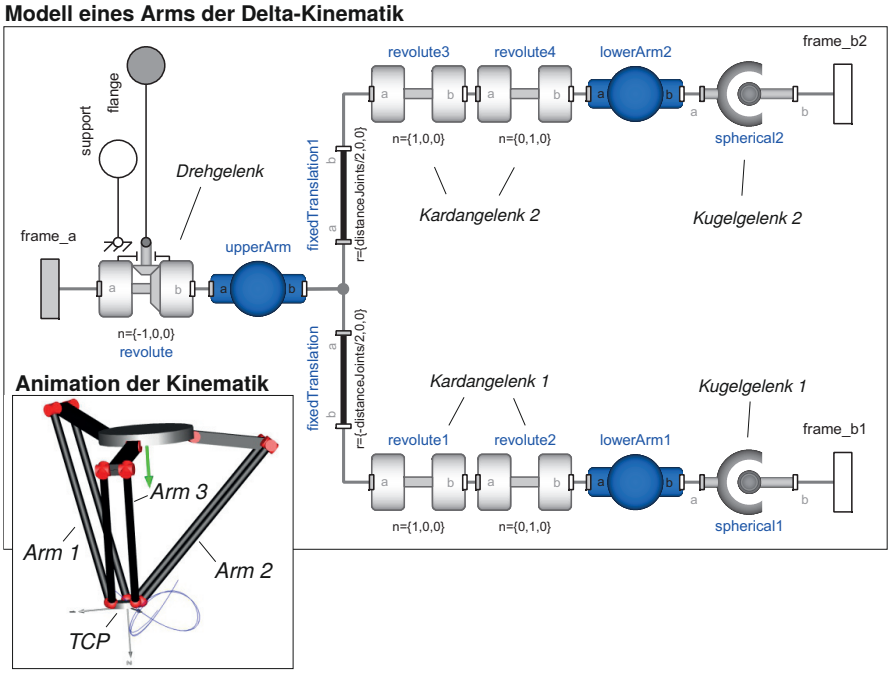


Bild 4-8: Modell eines Arms der Delta-Kinematik

Wissen über die kinematischen Zusammenhänge notwendig. Dieses Lösungswissen ist Bestandteil des LM-Modells, das eine bedarfsgerechte Parametrierung der Armlängen auf Basis des benötigten Arbeitsraums ermöglicht. Clavel gibt hierzu Werte für die Verhältnisse der geometrischen Größen an, die in einem guten Kompromiss zwischen der Größe des Arbeitsraums und den dynamischen Eigenschaften resultieren [Cla91, S. 77]. Demnach soll für „Standard Delta-Roboter“ gelten:

$$(R_A - R_B) = 0{,}63 \cdot L_A$$
$$L_B = 2 \cdot L_A$$

Hierbei stehen L_A und L_B für die Längen von Ober- und Unterarm und R_A für den Radius der Grundplatte. Mit R_B ist der Radius gemeint, auf dem sich die Mittelpunkte der Verbindung zwischen den beiden Endeffektor-Koppelpunkten der Unterarme befinden (vgl. Bild 4-7). Auf Basis dieser Verhältnisse und der Angabe der Größe des Endeffektors (und damit von R_B) wird eine überschlägige Parametrierung der Kinematik mithilfe angegebener Werte für Länge und Breite

einer rechteckigen Arbeitsfläche möglich. Dazu werden maximale Armwinkel in den Grenzen des Arbeitsraums angenommen, welche es der Delta-Kinematik ermöglichen, auch hier noch die nötigen Kräfte aufzubringen. Optional können die so ermittelten Parameter als Anfangswerte für eine Optimierung der geometrischen Größen mittels mathematischer Optimierungsalgorithmen genutzt werden. Dies wurde exemplarisch unter Nutzung der Modelica *Optimization Library* [Pfe12] umgesetzt. Dazu werden die nötigen Modelica-Funktionen und Einstellungen hinterlegt, um diejenigen Armlängen und Radien zu ermitteln, welche die *Manipulierbarkeit* der Delta-Kinematik in den beschriebenen Grenzen des Arbeitsraums zu einem Maximum führen. Die Manipulierbarkeit w einer Roboter-Kinematik ist nach Yoshikawa [Yos85] definiert als Funktion der Jakobi-Matrix $\underline{J}(\underline{q})$:

$$w = \sqrt{\det \underline{J}(\underline{q})\underline{J}^T(\underline{q})}\,, \text{ mit } \dot{\underline{r}}_{TCP} = \underline{J}(\underline{q}) \cdot \dot{\underline{q}}$$

Als einziges Gütekriterium J_w fungiert in diesem Fall die gemittelte Summe der quadrierten Manipulierbarkeiten für eine Anzahl n von Punkten im gegebenen Arbeitsraum:

$$J_w = \frac{1}{n} \cdot \sum_{i=1}^{n} w_i^2 = \frac{1}{n} \cdot \sum_{i=1}^{n} \det \underline{J}(\underline{q}_i)\underline{J}^T(\underline{q}_i) \tag{4-5}$$

Weitere Zielfunktionen und Methoden zur Optimierung der geometrischen Größenverhältnisse werden u. a. in [KGK03] vorgestellt.

Unter der Annahme zylindrischer bzw. kubischer Stäbe und der Dichte gängiger Werkstoffe werden weiterhin die Massen und Trägheitstensoren der zur Delta-Kinematik zusammengesetzten Starrkörper ermittelt. Folglich reicht bereits die Angabe folgender „Minimalparameter“ für eine realistische Simulation aus:

- Breite des Arbeitsraums b_{WS},
- Länge des Arbeitsraums l_{WS},
- Endeffektorradius r_E.

Das zusammengesetzte Lösungsmustermodell „Delta-Roboter“

Analog zur Kaskade der Lösungselemente (vgl. Bild 2-14 auf Seite 48) können selbstverständlich auch Lösungsmuster zu neuen, höherwertigen Lösungsmustern zusammengesetzt werden. Im Falle der kooperierenden Delta-Roboter gilt es, das Lösungsmuster *Delta-Kinematik* über jeweils ein Getriebe mit dem Lösungsmuster *Servoantrieb* zu kombinieren. Durch die Instanziierung und Verbindung der einzelnen Aspekte/Modelle der Lösungsmuster entsteht das neue System

„Delta-Roboter“ (vgl. Bild 3-5 auf Seite 68). Die Modelle müssen folglich nicht von Neuem aufgebaut, sondern lediglich an den Stellen ergänzt werden, an denen bisher noch kein (adäquates) Lösungsmuster verfügbar ist. Dadurch kann zum einen in erheblichem Maße Modellierungsaufwand eingespart, zum anderen das Risiko von Fehlern in der Modellierung jedes einzelnen der Lösungsmusteraspekte deutlich verringert werden.

Für das idealisierte Dynamikmodell bedeutet dies, dass nun lediglich ein Modell der Getriebe erstellt werden muss, da sowohl das Antriebsmodell als auch das MKS-Modell der Delta-Kinematik (inkl. inverser Kinematik) wiederverwendet werden können. Das Getriebemodell beinhaltet im einfachsten Fall lediglich das Übersetzungsverhältnis $\ddot{u}$ ($\Gamma_{Getriebe} = 1,\ n_{min,Getriebe} = 2$). Mithilfe des zusammengesetzten Modells (vgl. Bild 3-6 auf Seite 71) kann dann die Funktionsfähigkeit des Systems auf die Probe gestellt werden. Bild 4-9 zeigt Simulationsergebnisse für einen Delta-Roboter mit den Parametern der Prinziplösung (Modellparameter siehe Tabelle 4-2). Um die Dynamik des Systems zu zeigen, wird eine hochfrequente sog. Lissajous-Figur für den TCP vorgegeben, die sich aus überlagerten Sinusschwingungen unterschiedlicher Frequenz und Amplitude für die einzelnen Endeffektor-Freiheitsgrade zusammensetzt (Bild 4-9 oben). Diese Soll-Vorgabe wird mittels der inversen Kinematik in die Gelenkkoordinaten umgerechnet und durch die Servoantriebe eingeregelt[6] (Bild 4-9 zeigt exemplarisch den Winkel für θ_1). Man kann erkennen, dass es aufgrund der sprunghaften Aufschaltung der Anregung ca. 0,1 s dauert, ehe der TCP der Vorgabe folgt. In dieser Zeit gelangt der Servoantrieb an die Stellgrößenbegrenzung für den zulässigen Strom (Bild 4-9 unten). Danach folgen die Ist-Werte den Soll-Werten für diese dynamischen Anregung sehr gut, auch wenn die Stellbegrenzung zeitweise beinahe erreicht wird.

Weiterhin zeigen die gestrichelt eingezeichneten Verläufe, dass mithilfe des Modells die Einflüsse unterschiedlicher Regelstrategien erprobt werden können. Bei identischen Reglerparametern wird hier auf eine Vorsteuerung der Soll-Winkelgeschwindigkeiten verzichtet, was zu einer deutlichen Verschlechterung führt. Die Simulationen zeigen dennoch die prinzipielle Funktionsfähigkeit des Systems, sodass auch das entstandene idealisierte Gesamtmodell als Bestandteil des neuen Lösungsmusters *Delta-Roboter* aufbereitet und zur Verfügung gestellt werden kann. Die Modellierungstiefe und die Modellkomplexität können dafür gemäß den Gleichungen 4-6 angegeben werden. Die Anzahl der Zustände und damit die Komplexität kann hier nicht durch das Aufsummieren der Bestandteile ermittelt werden, denn die drei Freiheitsgrade der Delta-Kinematik sind mit den rotatorischen Freiheitsgraden der Antriebe und Getriebe fest verkoppelt. Somit ergeben sich insgesamt 12 Zustände (6 Zustände durch die Kinematik und jeweils noch 2 verbleibende Zustände aufgrund der Antriebe). Diese Kopplung wird bei der Er-

[6] Regelung in Gelenkkoordinaten

Tabelle 4-2: Modellparameter des zusammengesetzten Modells

Teilmodell	Parameter	Wert
Delta-Kinematik	Endeffektorradius r_E	0,0721 m
	Masse des Endeffektors m_E	0,1635 kg
	Radius der Grundplatte R_A	0,2022 m
	Länge des Oberarms L_A	0,32 m
	Masse des Oberarms m_A	0,1792 kg
	Länge des Unterarms L_B	0,8495 m
	Masse des Unterarms m_B	0,1156 kg
Servoantriebe	Nenndrehmoment M_N	0,9 N m
	Maximalmoment M_{Max}	7 N m
	Drehmomentkonstante K_M	0,6 N m/A
	Nenndrehzahl ω_N	8000 1/min
	Versorgungsspannung U_{Net}	400 V
Getriebe	Übersetzungsverhältnis $\ddot{u}$	32
Sinusanregung	Frequenz	[3 Hz 6 Hz 8 Hz]
	Amplitude	[0,2 m 0,12 m 0,09 m]
	Phase	[0° 45° 0°]

mittlung der Gewichtungsfaktoren zur Berechnung der Gesamtmodellierungstiefe nicht berücksichtigt (siehe Gleichung 4-3 auf Seite 92). Hier geht die Summe der Minimalzustände $\sum_{i=1}^{k} n_{min,i} = 3 \cdot 2 + 6 + 3 \cdot 2 = 18$ ein. Die jeweils identischen Modelle für die drei Antriebe und Getriebe werden zusammengefasst.

$$
\begin{aligned}
K_{DeltaRoboter} &= n_{DeltaRoboter} = 12 \\
\Gamma_{DeltaRoboter} &= \gamma_{Antriebe}\,\Gamma_{Antrieb} + \gamma_{Kinematik}\,\Gamma_{Kinematik} + \gamma_{Getriebe}\,\Gamma_{Getriebe} \\
&= \gamma_{Antriebe} \cdot 2 + \gamma_{Kinematik} \cdot 2 + \gamma_{Getriebe} \cdot 1 = 1,64
\end{aligned}
$$

$$
\begin{aligned}
\text{mit } \gamma_{Antriebe} &= 3 \cdot \frac{1+2}{7+18} = \frac{9}{25}, \\
\gamma_{Kinematik} &= \frac{1+6}{7+18} = \frac{7}{25}, \\
\gamma_{Getriebe} &= 3 \cdot \frac{1+2}{7+18} = \frac{9}{25},
\end{aligned}
\tag{4-6}
$$

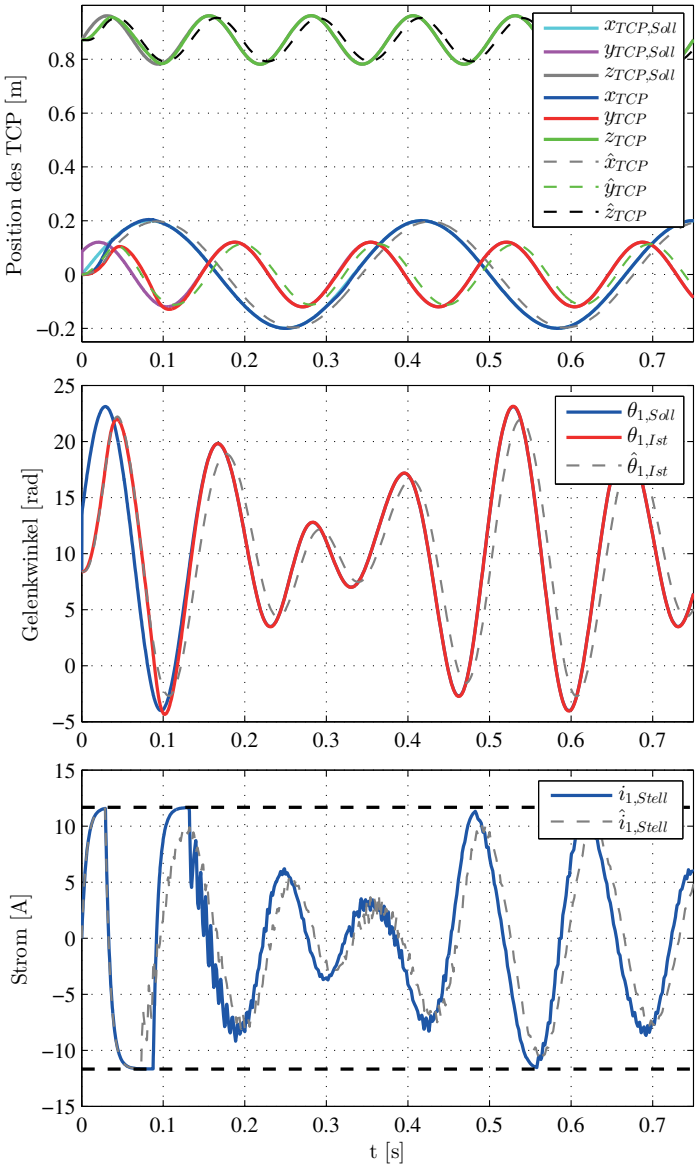


Bild 4-9: Simulationsergebnisse für eine Lissajous-Figur. Die durch ein ˆ gekennzeichneten Größen stellen den Fall ohne Vorsteuerung der Soll-Geschwindigkeiten in der Antriebsregelung dar.

4.3.2 Idealisierte Modellierung von Kontakten

Die bislang beschriebenen Modelle werden direkt einem Lösungsprinzip sowie den Systemelementen zugeordnet, die es nutzen. Daneben gibt es allerdings auch Modelle, die den Energiefluss zwischen zwei Systemelementen darstellen. Ein Beispiel hierfür sind solche zur Abbildung der Kontaktvorgänge zwischen der Schlägerplatte der Delta-Roboter und dem Spielobjekt. Zwar beinhalten Mehrkörpersimulationsmodelle in der Regel deutlich weniger solcher „losen“ als feste kinematische Verbindungen, dennoch stellt die adäquate Modellierung von Kontakten einen Schlüsselfaktor und eine besondere Herausforderung dar. Folglich kommt es hier vor allem auf die Wahl der richtigen Modellierungstiefe an, um den angemessenen Kompromiss zwischen nötiger Genauigkeit und Rechenaufwand zu finden [Juh08]. Dies trifft im Besonderen auf den Kontext der frühzeitigen Dynamikanalysen zu, sowohl was die Nutzung der Kontaktmodelle als auch was die Interpretierbarkeit der Ergebnisse angeht. Daher wurde für diesen Anwendungsfall die Modelica-Bibliothek *IdealizedContact* zur idealisierten Modellierung von Kontakten aufgebaut (vgl. [OWT14]). Sie basiert u. a. auf den studentischen Arbeiten von Wang [Wan12, Wan11] und Gringard [Gri14]. Die Modellierung der Kontaktmechanik ist Bestandteil vieler aktueller Forschungsarbeiten auch im Bereich MKS-Simulation [Hip04]. Zwar beschreiben Otter et al. Möglichkeiten zur Umsetzung von Kontaktmodellen in Modelica [OEL05], dennoch ist die entstandene Bibliothek *IdealizedContact* die erste verfügbare ihrer Art in Modelica.

Modellierungsziele und Abgrenzung zur Umwelt

Das Ziel ist es, das Wissen über die Modellierung von Kontakten in einer Modellbibliothek zu speichern. Die Kontaktmodellierung soll sich nahtlos in das Konzept der komponentenweise Wiederverwendung von Dynamikmodellen eingliedern. Die besondere Schwierigkeit besteht hierbei darin, dass Kontaktvorgänge und vor allem die Kraftangriffspunkte von der geometrischen Form der Kontaktpartner abhängen. Das bedeutet, dass ein Kontaktmodell nicht unabhängig von den betreffenden Komponenten definiert werden kann. Darüber hinaus wird bei einem MKS-Modell in der Regel mit Starrkörpern gerechnet, die durch ihren Schwerpunkt repräsentiert werden. Die Form der Körper wird bei der Berechnung zumeist gänzlich vernachlässigt und nur zu deren Visualisierung angezeigt. Dies trifft auch für die Umgebung Dymola/Modelica zu, für welche die Bibliothek *IdealizedContact* zur idealisierten Modellierung von Kontaktvorgängen aufgebaut wird. Nutzer dieser Bibliothek sollen durch sie in die Lage versetzt werden, die eingangs beschriebenen Dynamikanalysen durchzuführen.

Man kann beobachten, dass Kontaktflächen in der industriellen Anwendung vornehmlich einfache Formen haben. So wird sichergestellt, dass der Kontaktvor-

gang reproduzierbar und verlässlich ist. Aus diesem Grund wird der Umfang der Bibliothek zunächst auf Kombinationen folgender, elementarer Kontaktflächen beschränkt:

- rechteckige und runde ebene Flächen,
- Zylindermantelflächen,
- Kugeloberflächen,
- konvexe, parametrierbare Mantelflächen (z. B. Ellipsoide).

Kontaktflächen stellen einen dünnen, masselosen Eindringkörper dar, der mithilfe der *Frame*-Schnittstelle der Modelica Standard Library (MSL) mit beliebigen Starrkörpermodellen verbunden werden kann. Seine Ausdehnung (bzw. Größe) in den jeweiligen Koordinatenrichtungen wird über Parameter festgelegt. Die Umsetzung und Funktionsweise der Modelle soll im Folgenden am Beispiel des Kontakts zwischen Schlägerplatte und Spielball der kooperierenden Delta-Roboter erläutert werden. Hierbei handelt es sich um die Kombination einer kugelförmigen Oberfläche mit der eines runden Planares.

Damit die Modelle des Balls und der Schlägerplatte auch in anderen Anwendungsfällen wiederverwendet werden können, ist es erforderlich, die Definition der potentiellen Kontaktoberflächen von der eigentlichen Kontaktberechnung zu trennen. Indem nur die benötigten Flächen und nicht der gesamte Körper als Kontaktpartner modelliert werden, wird darüber hinaus unnötiger Rechenaufwand vermieden. Die Kontaktberechnung erfordert jedoch Informationen über die Form der Kontaktpartner, welche über die konventionelle *Frame*-Schnittstelle nicht zur Verfügung stehen. Daher wird eine neue, erweiterte Schnittstelle eingeführt (vgl. Quellcode 4-3), die zusätzlich den Oberflächentyp (Zeile 3) sowie einen Vektor (12. Ordnung, Zeile 4) beinhaltet. Letzterer gibt die Dimensionen (z. B. Länge und Breite oder den Radius) und die zugehörigen Richtungsvektoren der Kontaktoberfläche im körperfesten Koordinatensystem an.

Quellcode 4-3: Definition der Kontaktschnittstelle

```
1 connector Contact_a "Contact interface of surface definition"
2  Modelica.Mechanics.MultiBody.Interfaces.Frame_a frame;
3  IdealizedContact.Interfaces.ShapeTypeOutput contactShape;
4  Modelica.Blocks.Interfaces.RealOutput u[12];
5 end Contact_a;
```

Im Beispiel der kooperierenden Delta-Roboter besitzt das zweimal instanziierte Modell der Schlägerplatte eine runde, planare Kontaktoberfläche (vgl. Bild 4-10). Der interne Aufbau des Ballmodells ist im rechten Teil des Bildes zu sehen. Der Block mit der Bezeichnung `sphericalBody` ist Bestandteil der Modelica Standard Library (MSL) und definiert u. a. die Masse. Er ist verbunden mit dem Block

`sphericalContactSurface` aus der Kontaktbibliothek, welcher die Kontaktoberflächendefinition beinhaltet. Dieser wird mithilfe der Kontaktschnittstelle mit je einem sog. „Kontaktblock“ verbunden. Der Kontaktblock wird jeweils zwischen die Schnittstellen zweier Kontaktpartner eingesetzt und entsprechend der vorliegenden Kombination („SphereToCircle“) parametriert. Letzteres erfolgt mithilfe der *replaceable*-Methode zur Klassenparametrierung [Mod12] und führt dazu, dass das jeweils erforderliche Modell zur Kontaktberechnung eingesetzt wird.

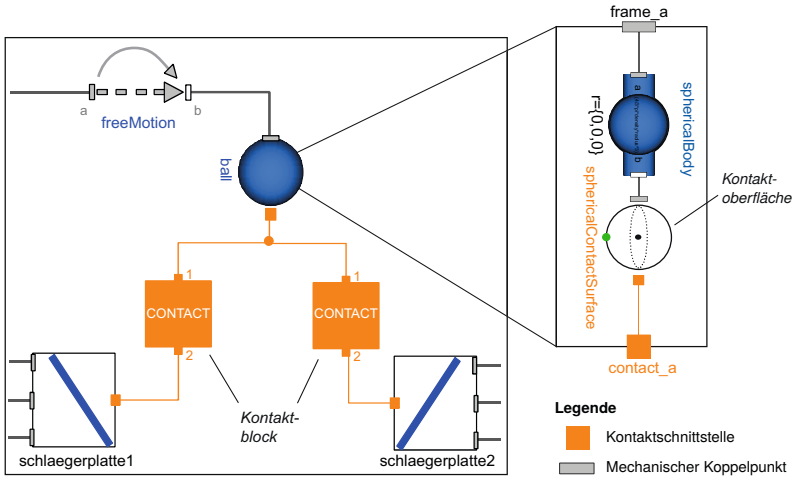


Bild 4-10: Modellierung des Kontakts zwischen den Schlägerplatten und dem Spielball [GTS14, S. 167]

Modellierung

Vor dem skizzierten Hintergrund und auf Grundlage der Bewertung gängiger Modellierungsansätze hinsichtlich Qualität, Effizienz, Komplexität, Robustheit und Flexibilität in [Hip04, S. 53] wird ein Modellierungsansatz auf Basis des sog. *Randschichtmodells* gewählt[7]. Dies entspricht der aus der Bauphysik bekannten *Winklerschen Bettung* (engl.: elastic foundation) und sieht die Approximation des eigentlich kontinuierlichen Spannungsbereichs durch Diskretisierung mithilfe einer Reihe von gleichartigen, voneinander unabhängigen Federn vor [Pop10]. Grundlegende Annahme ist weiterhin, dass das Kontaktgebiet klein gegenüber den Abmessungen der Kontaktpartner ist. Die makroskopischen Massen- und Trägheitseigenschaften des dynamischen Systems können dann entkoppelt von

[7] Neben diesem kraftbasierten Ansatz gibt es zum Beispiel ebenso Ansätze, die auf der Auswertung von Impulsbilanzen beruhen.

den Kontaktkräften betrachtet werden (Separation der elastischen Eigenschaften und der Trägheitseigenschaften), welche als Oberflächenkräfte eingeleitet werden können (vgl. [Pop10, S. 317]). Die Approximation des Kontaktgebiets wird umso genauer, je feiner diskretisiert wird. Für jede Feder müssen jedoch die Kontaktpunkte auf den beiden Oberflächen und die Eindringtiefe p bestimmt werden. Unter Berücksichtigung der Modellierungsziele ist hier lediglich das makroskopische Verhalten und nicht etwa die inneren Spannungen oder die Verformung der Körper von Interesse. Aus diesem Grund wird die Anzahl der Federn bzw. Kontaktpunkte so weit wie möglich reduziert, sodass zum Beispiel bei Paarungen, die Kugeln beinhalten, nur ein Kontaktpunkt berechnet werden muss (vgl. Bild 4-11).

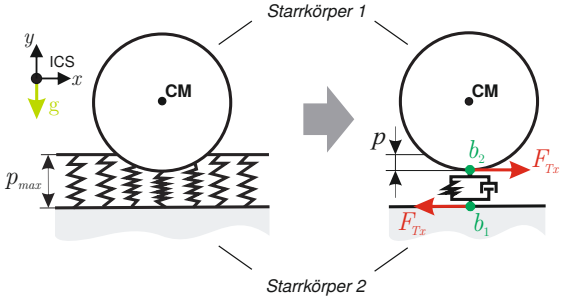


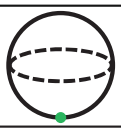
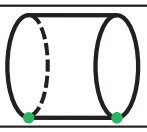
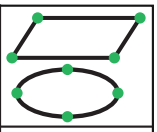
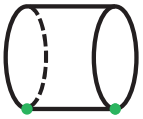
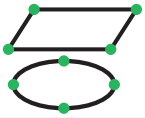
Bild 4-11: Idealisierung und Erweiterung des Randschichtmodells (ebene Darstellung, vgl. [OWT14])

Das Randschichtmodell berücksichtigt – wie viele andere Ansätze in der Literatur auch – nicht die dämpfenden Effekte, die abhängig von den Materialien und insbesondere bei hohen Kontaktgeschwindigkeiten ins Gewicht fallen [Hip04]. Daher wird parallel zu jeder Feder ein Dämpferelement eingesetzt. Folglich ist zur Bestimmung der Normalkraft F_N zusätzlich die Eindringgeschwindigkeit $\dot{p}$ notwendig. Mithilfe der Normalkraft kann anschließend die Tangentialkraft F_T ermittelt werden. Die Bestimmung der Kontaktpunkte $b_{1,2}$ sowie die jeweils zugrundeliegenden Kraftgesetze werden im Folgenden erläutert.

Bevor die Kontaktkraft zwischen den beiden Starrkörpern eingeprägt werden kann, müssen zunächst die potentiellen Kontaktpunkte auf den beiden Kontaktpartnern bestimmt werden. Tabelle 4-3 zeigt eine Übersicht über deren Anzahl für die verschiedenen Kombinationen der Kontaktflächen. Die Berechnung der potentiellen Kontaktpunkte kann grundsätzlich auf zwei verschiedene Arten und Weisen geschehen. Zum einen können bei einfachen Geometrien elementar-geometrische Beziehungen ausgenutzt werden, um die Punkte mit dem geringsten Abstand zu ermitteln (siehe dazu [OWT14]). Andererseits können sie mithilfe numerischer Abstandsminimierung gewonnen werden. Letzteres wurde in [Gri14] für konvexe,

parametrische Körper mit punktförmigem Kontaktgebiet umgesetzt. Beiden Methoden gemein ist, dass die Berechnung der potentiellen Kontaktpunkte in jedem Simulationsschritt durchgeführt wird. Daher steigt die Rechenzeit mit der Anzahl der nötigen Kontaktpunkte. Im Folgenden wird diese Kontaktpunkterkennung exemplarisch für das Beispiel erläutert, sie basiert auf [Wan11] und [Wan12].

Tabelle 4-3: Übersicht der Approximation des Kontaktgebiets und Anzahl potentieller Kontaktpunkte [OWT14]

			
	**punktförmig** (1 pot. K.P.)	**punktförmig** (1 pot. K.P.)	**punktförmig** (1 pot. K.P.)
	**punktförmig** (1 pot. K.P.)	**punktförmig/ linear** (1/2 pot. K.P.)	**linear** (2 pot. K.P.)
	**punktförmig** (1 pot. K.P.)	**linear** (2 pot. K.P.)	**planar** (4/5 pot. K.P.)

Kontaktpunkterkennung: Die Kontaktpunkterkennung beim Kontakt zwischen Ball und Schlägerplatte ist vergleichsweise einfach. Bild 4-12 zeigt je ein Beispiel für zwei unterschiedliche Fälle. Durch das *Frame* in der Kontaktschnittstelle sind die Schwerpunkte der beiden Körper a_1 und a_2 sowie ihr jeweiliges körperfestes Koordinatensystem (Body Coordinate System (BCS)) verfügbar. Zusätzlich existieren aufgrund der erweiterten Kontaktschnittstelle Informationen über die Radien R und r von Schlägerplatte und Spielball. Im Fall I kann der erste Kontaktpunkt b_1 direkt bestimmt werden, indem der Schwerpunkt a_2 mithilfe der Matrix $\underline{P}$ auf die Ebene projiziert wird. Die Verschiebung bzgl. des Schwerpunkts a_1 ergibt sich dann durch:

$$\underline{r}_{1,p} = \underline{P} \cdot \underline{a}_2 = \begin{bmatrix} \underline{e}_{x,BCS1} & \underline{0} & \underline{e}_{z,BCS1} \end{bmatrix} \cdot \underline{a}_{2,BCS1}$$

Hierbei stehen die Vektoren $\underline{e}$ jeweils für die entsprechenden kanonischen Einheitsvektoren der angegebenen Basis. Im Fall II muss die Länge des Richtungsvektors

$\underline{r}_1$ jedoch auf den Radius R begrenzt werden. Allgemein kann folgende Vorschrift für beide Fälle aufgestellt werden:

$$\underline{r}_1 = \frac{\min(R, \left\|\underline{r}_{1,p}\right\|)}{\left\|\underline{r}_{1,p}\right\|} \cdot \underline{r}_{1,p}$$

Mithilfe des Kugelradius r lässt sich anschließend der Vektor $\underline{r}_2$ zum Erhalt von b_2 bestimmen.

$$\underline{r}_2 = \frac{r}{\left\|\overrightarrow{a_2 b_1}\right\|} \cdot \overrightarrow{a_2 b_1}$$

Die beiden Punkte b_1 und b_2 heißen potentielle Kontaktpunkte, da eine Kollision erst noch durch die Kontaktbedingung geprüft werden muss. Sie werden dazu kontinuierlich und zu jedem Zeitpunkt der Simulation berechnet. Bei anderen Kontaktflächenpaarungen (siehe Tabelle 4-3) ist in einigen Fällen zusätzlich eine Verschiebung der potentiellen Kontaktpunkt notwendig (vgl. [OWT14]). Im Falle des Kontakts wird bei ihnen die Kontaktkraft $\underline{F}_K$ eingeprägt.

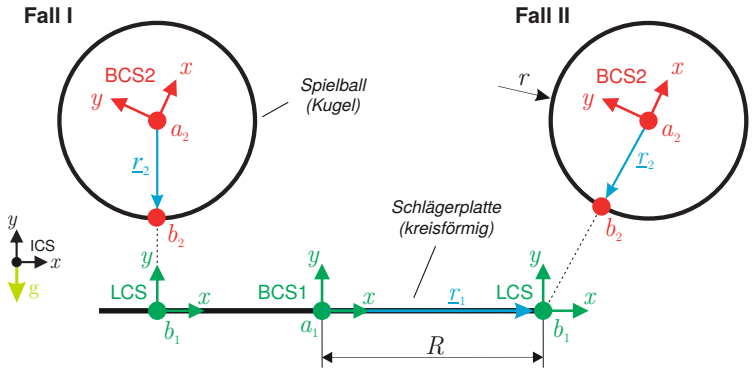


Bild 4-12: Detektion der potentiellen Kontaktpunkte auf Spielball und Schlägerplatte (ebene Darstellung, nach [Wan12, S. 36])

Modellierung der Kontaktkraft: Ein Kontakt liegt vor, wenn sich die Körper durchdringen. Dabei ist unerheblich welcher Körper in welchen eindringt. Die Eindringtiefe wird mit p bezeichnet (vgl. auch Bild 4-11). Im Beispiel lautet die Bedingung für die Kontaktkraft:

$$\begin{aligned} contact &= (\left\|\underline{P} \cdot \overrightarrow{a_1 b_2}\right\| \leq R) \wedge (0 \leq p \leq p_{max}) \text{ mit} \\ p &= -\underline{e}_{y,BCS1}{}^T \cdot \overrightarrow{b_1 b_2} \end{aligned} \tag{4-7}$$

Für diese Kontaktbedingung wird der Parameter maximale Eindringtiefe p_{max} eingeführt. Dadurch wird verhindert, dass ein weit unterhalb der Schlägerplatte befindlicher Spielball eine entsprechend große Kraft erfährt und von unten durch die Platte katapultiert wird. Bei der Berechnung der Kontaktkraft wird darauf geachtet, keine weiteren ereignisdiskreten Umschaltungen zu verwenden, die zu erhöhten Rechenzeiten aufgrund der Verringerung der Schrittweite durch die Schrittweitensteuerung des Lösers führen würden. Für den Normalkraftanteil der Kontaktkraft wird unabhängig von der Kontaktpaarung ein nichtlineares Feder-Dämpfergesetz mit den Konstanten c und d nach Lankarani und Nikravesh angenommen [LN94]:

$$F_N = c \cdot p^n + d \cdot p^m \cdot \dot{p}$$

Dies stellt ein kontinuierliches Modell mit Dämpfungshysterese dar. Im Gegensatz zu linearen Ansätzen treten hierbei keine Effekte ähnlich der Adhäsion auf, bei denen der kollidierte Körper haften bleibt (für $p = 0$ und $\dot{p} \neq 0$). Dennoch kann durch Parametrierung der Exponenten mit $n = 1$ und $m = 0$ ein lineares Kelvin-Voigt-Modell eingesetzt werden. Wählt man $n = m$, ergibt sich eine Formulierung analog zu Hunt und Crossley [HC75]. Die tangentialen Reibungsanteile der Kontaktkraft sind abhängig von der Normalkraft und der Relativgeschwindigkeit v_{rel} der Kontaktpartner, die von tangentialer oder rotatorischer Bewegung herrühren kann. Um auch hier diskrete Umschaltungen zu vermeiden, wird die stetig differenzierbare Approximation der Stribeck-Reibungskurve mittels hyperbolischer Tangensfunktionen nach Makkar et al. verwendet [MDS+05]. Demnach berechnet sich der Reibungsbeiwert μ_i durch

$$\mu_i(v_{rel,i}) = \gamma_1(\tanh(\gamma_2 v_{rel,i}) - \tanh(\gamma_3 v_{rel,i})) + \gamma_4 \tanh(\gamma_5 v_{rel,i}) + \gamma_6 v_{rel,i}.$$

Hierbei steht der Index i für die jeweilige Richtung der Schlägerplattenebene. Dieses kontinuierliche Modell hat neben den genannten Vorzügen auch Vorteile bei etwaiger Reglerauslegung [MDS+05], kann jedoch keine ideal haftende Reibkraft darstellen, da der Zustand des idealen Haftens unabhängig von der Relativgeschwindigkeit ist. Haften wird vielmehr durch ein Gleiten mit sehr kleinen Relativgeschwindigkeiten abgebildet, was allerdings den tatsächlichen Bedingungen in vielen tribologischen Systemen sehr nahe kommt [Pop10, S. 310].

Die gesamte Kontaktkraft $\underline{F}_K$ ergibt sich bzgl. des lokalen Koordinatensystems LCS zu:

$$\underline{F}_K = \begin{pmatrix} F_{Tx} \\ F_N \\ F_{Tz} \end{pmatrix} = \begin{pmatrix} \mu_x \cdot F_N \\ F_N \\ \mu_z \cdot F_N \end{pmatrix}$$

Parametrierung und Simulationsergebnisse

Um eine einfachere Parametrierung des Reibungsmodells im Sinne der Forderung aus Abschnitt 4.2 zu gewährleisten, können die unbekannten und nicht physikalischen Konstanten $\gamma_{1..6}$ mithilfe von Berechnungsvorschriften aus fünf Parametern gewonnen werden (vgl. Bild 4-13). Mit μ_s und μ_k werden die Gleit- und Haftreibungskoeffizienten angegeben. Die beiden Grenzgeschwindigkeiten $v_{\varepsilon 1}$ und $v_{\varepsilon 2}$ beschreiben die Anfänge der Misch- und Gleitreibungszone. Durch den Parameter k_v kann viskose Reibung modelliert werden. Ein Nachteil des gewählten kraftbasierten Ansatzes ist, dass die Steifigkeits- und Dämpfungsgrößen des Kontakts schwer abschätzbar sind. Sie hängen sowohl von den Materialpaarungen als auch von der Form und Größe beider Kontaktpartner ab. Für detaillierte Untersuchungen ist im konkreten Fall daher eine Parameteridentifikation unumgänglich. Die in der entstandenen Bibliothek enthaltenen Beispiele geben jedoch Hinweise zu den Größenordnungen. Zudem lassen die Angaben in [SGM08, S. 91] und [Bee94, S. 15] darauf schließen, dass bei linear-elastischen Werkstoffen für das Verhältnis von Dämpfungs- und Steifigkeitskoeffizient gilt:

$$0,001 \leq \frac{d}{c} \leq 0,1$$

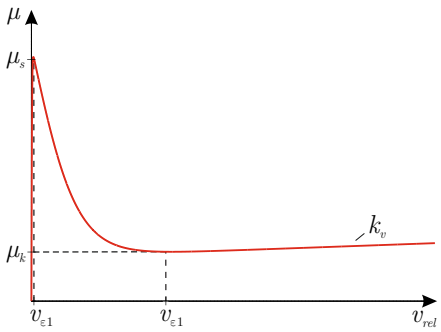


Bild 4-13: Approximation der Stribeck-Reibungskurve [OWT14])

Bild 4-14 zeigt Simulationsergebnisse für eine Beispielkonfiguration von Spielball und Schlägerplatte (Parameter siehe Tabelle 4-4). Der rotierende Spielball fällt hier exzentrisch auf die leicht um die x-Achse geneigte Schlägerplatte. Er springt zweimal auf, wobei jeweils der Kontakt erkannt und eine Normalkraft gestellt wird. Diese ist nichtlinear, beinhaltet Dämpfung und erreicht ein Maximum von $F_N = 1{,}16 \cdot 10^{-3}\,\mathrm{N}$. Anschließend verlässt der Ball die Platte und fällt herunter, da die Kontaktbedingung (Gleichung 4-7) nicht mehr erfüllt ist. Die tangentiale

Reibungskraft ist in diesem Beispiel vernachlässigbar klein. Weitere Simulationsergebnisse und Informationen zur Funktionsweise der Kontaktmodellierung finden sich in der genannten Literatur [OWT14, Gri14, Wan12, Wan11].

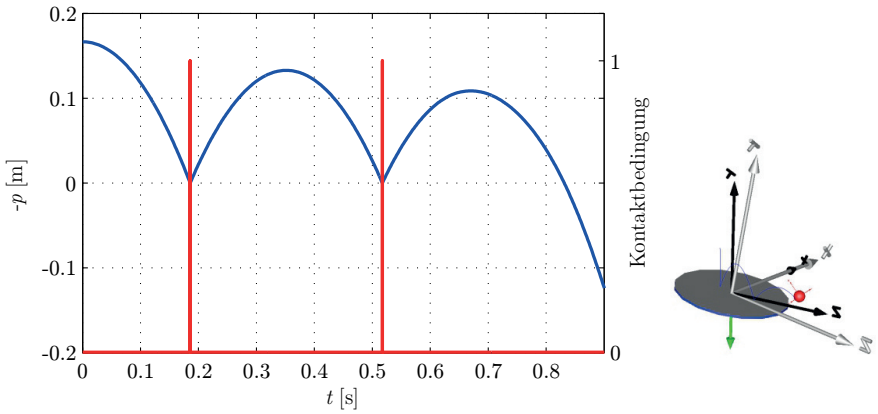


Bild 4-14: Simulationsergebnisse für die Eindringtiefe p (blau) und die Kontaktbedingung (rot) beim Fallen eines rotierenden Balls auf eine leicht geneigte Schlägerplatte

Tabelle 4-4: Parameter der Kontaktsimulation

Teilmodell	Parameter	Wert
Spielball	Radius r	21,75 mm
	Masse m_{Ball}	24 g
	initiale Rotationsgeschwindigkeit	[10 rad/s 2 rad/s 0 rad/s]
Schlägerplatte	Radius R	0,25 m
	Neigung	[10° 0° 0°]
Kontaktblock	Kontaktsteifigkeit c	$1 \cdot 10^{12}$ N/m
	Kontaktdämpfung d	$1 \cdot 10^{11}$ N s/m
	Steifigkeitsexponent n	1,5
	Eindringungsexponent m	1,5

4.4 Modelle in der Ausarbeitung

Die bisher beschriebenen Modelle dienten vornehmlich der Absicherung der prinzipiellen Funktionsfähigkeit im Rahmen der Konzipierungsphase. Im Laufe der

weiteren Entwicklung wird das mechatronische System ausgearbeitet und detailliert. Selbiges trifft auch auf die Dynamikmodelle des Systems zu, denn während des ganzheitlichen Entwurfs soll die Systemabstimmung unter möglichst realen Bedingungen analysiert und weiter optimiert werden. Dies setzt einen deutlich höheren Detaillierungsgrad voraus als in der Systementwurfsphase und erfordert spezifisches, detailliertes Wissen über die eingesetzten Komponenten und Systemelemente. Dieses Wissen kann zum Beispiel mithilfe von Messungen an einem Prüfstand oder Prüfmuster gewonnen werden. Aber auch sehr detaillierte Simulationsmodelle, wie sie zur Optimierung der Komponenten verwendet werden (Modellierungstiefe $\Gamma = 4$), können Aufschluss geben. Vor dem Hintergrund von Systemsimulationen sind diese Modelle in der Regel jedoch nicht oder zumindest nicht für alle Systemelemente nutzbar, da der Rechenaufwand aufgrund zu hoher Komplexität nicht praktikabel ist. Mit dem gewonnenen Wissen kann jedoch zielgerichtet abstrahiert und vereinfacht werden, sodass Modelle entstehen, die der Ziel-Modellierungstiefe $\Gamma^* = 3$ entsprechen (vgl. [Jus14a]). Der beschriebene Prozess ist vielfach mit sehr großem Aufwand verbunden, weshalb es auch hier sinnvoll ist, das erworbene Wissen als Lösungswissen in Form von Modellen abzulegen und wiederverwendbar zu machen. Der nachfolgende Abschnitt behandelt solche Modelle, die direkt einem spezifischen Lösungselement zuzuordnen sind, ehe abschließend ein Beispiel für die Aufbereitung weiterer Modelle der Ausarbeitung gegeben wird.

4.4.1 Lösungselementmodelle für die detaillierte Ausarbeitung und Analyse

In der fachspezifischen Ausarbeitung werden konkrete, fertige (Teil-)Lösungen ausgewählt, die in neuen Systemen ohne große Anpassung wiederverwendet werden können (vgl. Kapitel 3). Dies umfasst Lösungen, die von Zulieferern bezogen werden, aber auch solche, die intern im Unternehmen zum Beispiel aus vorherigen Projekten vorliegen. Sie werden mit dem Begriff Lösungselement bezeichnet. Unter dem Begriff *Lösungselementmodell* werden Dynamikmodelle der Modellierungstiefe $\Gamma \approx 3$ verstanden, die das spezifische Verhalten eines Lösungselements detailliert wiedergeben. Lösungselemente stellen konkrete Ausprägungen von Lösungsmustern dar. Das LM-Modell eines Systemelements wird folglich für den Fall, dass ein bestehendes Lösungselement ausgewählt wurde, ersetzt. Damit die Änderung des Detaillierungsgrads erfolgen kann, ohne die Struktur des Modells zu verändern, ist es erforderlich, dass die Schnittstellen übereinstimmen. Darüber hinaus kann es für viele Lösungselemente sinnvoll sein, verschiedene Detaillierungsgrade vorzuhalten, da der Einsatzzweck und die damit verbundenen Genauigkeitsanforderungen variieren können. Um in beiden Fällen einen Austausch zu vereinfachen, werden

die durch das Interface-Modell (siehe Bild 4-5) definierten Minimalschnittstellen auch an zugehörige Lösungselementmodelle weitervererbt (vgl. auch Bild 4-15).

Da die Erstellung von Lösungselementmodellen Aufwand und Detailwissen erfordert, ist es sinnvoll sie wiederzuverwenden. Hersteller von Lösungselementen können an dieser Stelle ansetzen und sich einen signifikanten Wettbewerbsvorteil erarbeiten, indem sie ihren Kunden Wissen über ihr Produkt in Form von Dynamikmodellen zur Verfügung stellen (vgl. [Kuf11, S. 47]), ähnlich wie es vielfach bereits mit CAD-Modellen gehandhabt wird. Auch hier geht das Angebot der Hersteller bereits seit langem über die rein textuellen Beschreibungen in Katalogen hinaus [ODB+12]. Dies bedeutet natürlich auch für den LE-Hersteller Mehraufwand. Aufgrund seiner großen Expertise auf dem Gebiet und des i. d. R. vorhandenen Bestands an Modellen und Messungen wird dieser allerdings deutlich geringer ausfallen als bei den meisten Kunden. Zudem wird er für ähnliche Produkte seines Portfolios – die derselben Lösungselementklasse zuzuordnen sind – zumeist dasselbe, produktspezifisch parametrierbare Modell verwenden können. Darüber hinaus sind herstellerübergreifende Kooperationen in Form von Modellen denkbar, die den Stand der Technik für ganze Lösungselementgruppen repräsentieren. Bild 4-15 zeigt ein mögliches Beispiel für ein solches herstellerübergreifendes Lösungselementmodell eines Synchron-Servoantriebs (vgl. [GTS14]), das in der Ausarbeitung sowohl der kooperierenden Delta-Roboter als auch des Teigkneters verwendet werden kann.

Hierin wird der mechanische Teil (`mechanical part`) des Antriebs durch eine Trägheit und durch viskose Lagerreibung wiedergegeben. Im elektrischen Teil (`electrical part`) finden sich die Induktivität sowie der ohmsche Widerstand. Da der Motor geregelt betrieben wird, sodass sich eine Stromkomponente zu Null einstellt ($I_d = 0$), kann das anliegende Drehfeld durch einen Gleichstrom approximiert werden [Ise08, S. 257]. Die korrekte Abbildung der Pulsweitenmodulation[8] im Frequenzumrichter würde zu sehr kleinen Simulationsschrittweiten führen. Ihr Einfluss wird daher auf eine Totzeit reduziert. Des Weiteren wird die in der Antriebstechnik standardmäßig verwendete Kaskadenregelung von Strom (`current control`) und Winkelgeschwindigkeit (`omega control`) mit PI(D)-Anti-WindUp-Reglern sowie des Winkels mit einem P-Regler (`phi control`) abgebildet (vgl. Abschnitt 4.3.1). Für die Sollwerte sind Sollwertfilter vorgesehen. Ein weiterer Eingang erlaubt eine externe Momentenvorsteuerung zusätzlich zur internen Vorsteuerung der Rotorträgheit. Die Begrenzung des Antriebsmoments wird im Beispiel durch ein variables Limit des Stromsollwerts realisiert, das wiederum mithilfe der üblicherweise zur Verfügung gestellten Kennlinie der Motor-Umrichter-Kombination bestimmt wird. Die Grenzen des Antriebs können so auf einfache

[8] Die Pulsweitenmodulation (PWM) wird in der Regel zur Kommutierung verwendet. Übliche Frequenzen liegen zwischen 2 und ca. 20 kHz.

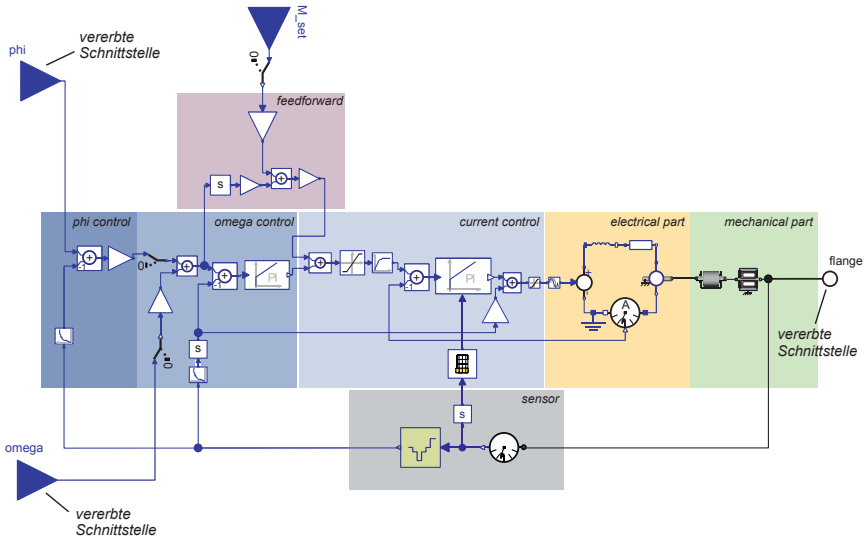


Bild 4-15: Beispiel eines herstellerübergreifenden Lösungselementmodells eines Servoantriebs in Dymola [GTS14, S. 105]

und schnelle Art und Weise auch außerhalb des Ankerstellbereichs wiedergegeben werden. Aufgrund der vielfach sehr genau vermessenen Kennlinien wird das nach außen sichtbare Verhalten detailliert wiedergegeben. Die Modellierungstiefe ist daher groß, auch wenn die Art der Modellierung wenig komplex ist. Ohne zu viel Rechen- und Modellierungsaufwand zu verursachen, gibt das Modell dadurch deutlich mehr Eigenschaften des realen Produkts detailliert wieder als das entsprechende Lösungsmuster-Modell (vgl. Abschnitt 4.1). Wichtig ist zudem, dass die vom Nutzer einzustellenden Parameter (bspw. die Reglerparameter) den realen Einstellmöglichkeiten des Lösungselements möglichst exakt entsprechen, sodass diese bei der späteren Inbetriebnahme übertragen werden können. Die Parameter des in Bild 4-15 gezeigten Modells lauten:

- Drehmomentkonstante K_M,
- maximaler Strom I_{Max},
- Versorgungsspannung U_{Net},
- Rotorträgheit J_R,
- konstantes Reibmoment M_R,
- Induktivität L_i,

- Widerstand R_i,
- PWM-Frequenz f_{PWM},
- Reglerparameter der aktiven Reglerkaskaden,
- Grenzkennlinie der Motor-Umrichter-Kombination.

Das hier beschriebenen Beispielmodell besitzt 8 Zustände und ist in der Lage spezifisches Verhalten abzubilden. Folglich ergeben sich im Gegensatz zum Lösungsmustermodell eines Servoantriebs (vgl. Bild 4-6 auf Seite 102) folgende Werte für die Komplexität und Modellierungstiefe:

$$K_{AntriebLE} = 8$$
$$\Gamma_{AntriebLE} = 3$$

In den Unternehmen sind neben unterschiedlichen Modellierungssprachen auch ganz verschiedene Werkzeuge im Einsatz. Dieser Umstand erschwert die Einbindung von Modellen anderer Unternehmen. Gleichzeitig haben Zulieferer oftmals berechtigte Hemmungen, vorhandenes Know-How in Form von White-Box-Modellen[9] offenzulegen. Diesen Problemen kann durch die Nutzung sogenannter FMUs (Functional Mock-Up Units) begegnet werden. Das FMI ist ein Austauschformat für Simulationsmodelle, welches mittlerweile von sehr vielen Softwarewerkzeugen unterstützt wird. Es wurde ursprünglich im Forschungsprojekt *Modelisar* entwickelt [FMI14], in dem sich neben Forschungseinrichtung (z. B. das Deutsches Luft- und Raumfahrtzentrum) und Automobilherstellern/-zulieferern (z. B. Daimler) auch Softwarehersteller (z. B. Dassault Systèmes) engagierten. Heute hat die Modelica Association die Weiterentwicklung und Verbesserung des Standards übernommen. Eine FMU besteht aus einer Archivdatei, welche neben dem eigentlichen Modell in Form von C-Code und/oder DLLs auch eine Modellbeschreibung im XML-Format und weitere Modelldaten (wie Tabellen oder Kennlinien) enthält. Aufgrund der Konvertierung nach C sind allerdings in der aktuellen Version 2.0 nur Signalfluss-Schnittstellen möglich.

4.4.2 Modellierung des Umfeldelements „Teig“

Im Anwendungsbeispiel des Teigkneters ist neben den einzelnen Systemelementen das Umfeldelement `Teig` von entscheidender Bedeutung, denn Teig zu bearbeiten bzw. herzustellen, stellt die Hauptfunktion des mechatronischen Systems dar (vgl. Bild 3-12 auf Seite 82). Im Sinne der Wiederverwendung wäre es erstrebenswert, den Teig in Gänze und umfassend für alle Zustände zu beschreiben, sodass

[9] Im Gegensatz zu Black-Box-Modellen bezeichnet man hiermit Modelle, deren Innenleben offen und damit frei zugänglich ist.

das Modell bei der Entwicklung beliebiger teigherstellender oder -verarbeitender Maschinen verwendbar wäre. Angesichts seiner sich ständig verändernden Eigenschaften und komplizierten biochemischen Zusammenhänge, die stark von den natürlichen Rohstoffen, ihrer Zusammensetzung und Qualität sowie von den Umgebungsbedingungen abhängen, erscheint dies jedoch nahezu unmöglich. Ein Dynamikmodell, das im Systemkontext verwendbar ist (vgl. Modellierungsziele für Modelle in der Ausarbeitung), kann daher nicht gänzlich unabhängig von den herrschenden Bedingungen im vorliegenden speziellen Kneter formuliert werden, was die Wiederverwendbarkeit natürlich einschränkt. Im Folgenden wird jedoch anhand des Beispiels beschrieben, wie Teilmodelle oder auch das Modell als Ganzes durch geeignete Strukturierung und Aufbereitung sinnvoll wiederverwendet werden können. Hierzu wird für den Teig eine Kombination aus physikalischer und empirischer Modellierung gewählt. Als Ausgangspunkt wird das Antriebsmoments der Knetspirale und die Teigtemperatur beim Knetprozess gemessen und analysiert. Beide Größen beinhalten charakteristische Informationen über den Teigzustand.

Modellierung des niederfrequenten Teigmoments: Übereinstimmend mit den Ergebnissen aus der Literatur [SCD+12] hat sich bei der Analyse der Messungen herausgestellt, dass die in den Teig eingebrachte, auf die Teigmasse m_d bezogene spezifische Energie E_s sich sehr gut als Bezugsgröße eignet. Denn dadurch ist das Modell unabhängig von den Betriebsdrehzahlen und der Teigmasse. Zunächst wird das Widerstandsmoment des Teiges M_d ermittelt, indem das gemessene Antriebsmoment um Verlustanteile bereinigt wird. Letztere werden anhand von Leerlaufmessungen modelliert und identifiziert. Durch Mitteln und Glätten kann ein Kennfeld bestimmt werden, das den nichtlinearen Zusammenhang zwischen der spezifischen Energie, der Ausgangstemperatur des Wassers T_w und dem niederfrequenten Teigmoment $M_{d\varphi}$ herstellt. Das Kennfeld ist Grundlage für ein kontinuierliches Modell des Teigs, welches das Teigmoment aus dem Motormoment der Antriebsspirale $M_{mot,S}$, Reibungsverlusten M_{loss} und der aktuellen Relativdrehzahl von Bottich und Spirale ω_{rel} berechnet:

$$M_{d\varphi} = m_d \cdot f(E_s, T_w) = M_{mot,S} - M_{loss} - M_{d\omega}$$
$$E_s = \frac{E_d}{m_d} = \frac{1}{m_d} \cdot \int_0^{t_{end}} M_d \cdot \omega_{rel} \, \mathrm{d}t$$

Modellierung hochfrequenter Anteile: Zur Modellierung hochfrequenter Anteile betrachtet man den Knetspalt, durch den der Teig bei jeder Umdrehung der Knetspirale gedrückt wird (vgl. Bild 4-16). Bei Teig handelt es sich rheologisch gesehen um ein visko-elastisches Material, das folglich sowohl viskose – d. h. dämpfende – als auch elastische Anteile besitzt [Mez10]. Das hier verwendete

lineare Modell für Teig ist das sog. Kelvin-Voigt-Modell, welches aus einer Feder und einem parallel geschalteten Dämpfer besteht. Dieses einfache Modell wird sehr häufig verwendet [Blo72, SS06] und ermöglicht in weiten Bereichen gute Übereinstimmungen zwischen den rheologischen Messungen am fertigen Teig und der Simulation. Jedoch verändern sich die visko-elastischen Eigenschaften des noch unfertigen Teigs im Laufe des Knetprozesses kontinuierlich. Daher werden die Parameter c_d und d_d des Modells ebenfalls in Abhängigkeit von der Bezugsgröße E_s ausgedrückt. Mithilfe des physikalischen Ersatzbildes (siehe Bild 4-16) kann die Differentialgleichung für das hochfrequente Teigmoment $M_{d\omega}$ aufgestellt werden. Aufgrund des mit der Spiraldrehfrequenz ω_S veränderlichen Spalts und des Hebels der Kraft F_y, oszilliert das Moment um einen Mittelwert und setzt sich aus überlagerten Schwingungen mit den Frequenzen ω_S und $2{\cdot}\omega_S$ zusammen (vgl. Gleichung 4-8) [Tra14]. Der Kraftanteil F_φ entsteht nicht durch die periodische Quetschung des Teigs und ist daher an dieser Stelle nicht von Belang.

$$\begin{aligned} M_{d\omega} &= F_y \cdot s = \{c_d(E_s) \cdot s + d_d(E_s) \cdot \dot{s}\} \cdot R_S \cos\varphi_S \\ &\quad \text{mit } s = s_{min} + R_S(1 - \sin\varphi_S) \\ &\quad \text{und } \varphi_S \approx \omega_S \cdot t \\ \Rightarrow M_{d\omega} &= c_d R_S(s_{min} + R_S)\cos(\omega_S \cdot t) - \frac{c_d R_S^2}{2}\sin(2\omega_S \cdot t) \\ &\quad - \frac{d_d R_S^2}{2}\omega_S(\cos(2\omega_S \cdot t) + 1) \end{aligned} \tag{4-8}$$

Modellierung der Teigtemperatur: Neben dem Widerstandsmoment ist die Temperatur des Teigs eine wichtige, charakteristische Größe für den Teigzustand. Um sie zu modellieren, wird eine Leistungsbilanz in Anlehnung an Shehzad et al. aufgestellt [SCD+12]. Die im Teig gespeicherte thermische Energie $E_{d,therm} = \int P_{d,therm}$ steigt mit der eingebrachten mechanischen Leistung $P_{d,mech} = M_d \cdot \omega_{rel}$ und verringert sich durch die an die Umgebung abgegebene Leistung $P_{d,loss}$. Letztere ist abhängig von der Temperaturdifferenz zwischen Teig (T_d) und Umgebung (T_{env}) sowie von der Oberfläche für den Wärmeaustausch. Es resultiert die folgende Differentialgleichung.

$$\begin{aligned} P_{d,therm} &= P_{d,mech,f} - P_{d,loss} \\ \Rightarrow m_d \cdot C_p \cdot \dot{T}_d &= P_{d,mech,f} - h_d \cdot S_d \cdot (T_d - T_{env}) \end{aligned} \tag{4-9}$$

Hierin ist C_p die spezifische Wärmekapazität des Teigs, h_d der Wärmeübergangskoeffizient für den Wärmeaustausch zwischen Teig und Umgebung und S_d die Teigoberfläche, welche sich aus der Füllhöhe und dem Bottichdurchmesser ergibt.

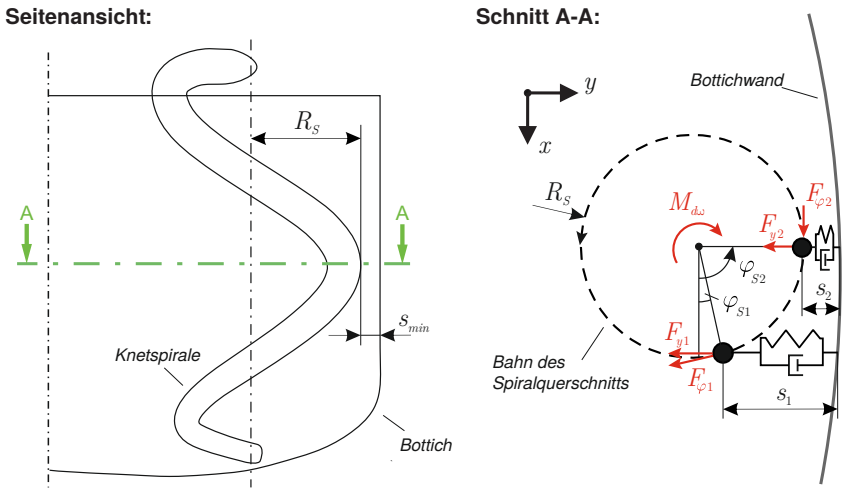


Bild 4-16: Physikalisches Ersatzbild zur Erstellung des Modells für das hochfrequente Teigmoment vgl. [Tra14, S. 41]

Da Änderungen der Temperatur nicht sprunghaft erfolgen können, wird die eingebrachte mechanische Leistung mittels eines Verzögerungsglieds erster Ordnung gefiltert. Aufgrund der sich ständig verändernden Eigenschaften des Mediums, zeigt sich dessen Zeitkonstante T_T jedoch als nicht konstant. Sie verringert sich deutlich im Laufe des Knetprozesses. Um diesen Effekt abzubilden, wird T_T ebenfalls in Abhängigkeit der Bezugsgröße E_s ausgedrückt. Hier wird ein linearer Zusammenhang angenommen. Somit berechnet sich der Anteil $P_{d,mech,f}$ der thermischen Leistung ($P_{d,therm}$), welcher durch das mechanische Drehmoment eingebracht wird, folgendermaßen:

$$P_{d,mech,f} = M_d \cdot \omega_{rel} - T_T(E_s) \cdot \dot{P}_{d,mech,f}$$

Hierdurch steht ein Modell zur Berechnung der Teigerwärmung auf Grundlage der mechanisch eingebrachten Leistung und der Umgebungstemperatur zur Verfügung. Dieses Modell setzt eine näherungsweise homogene Masse mit annähernd gleichförmiger Temperaturverteilung voraus. Hiervon kann allerdings erst ausgegangen werden, wenn die verschiedenen Zutaten, die zudem sehr unterschiedliche Ausgangstemperaturen haben, vermischt sind. Dafür muss sich die Knetspirale bei herkömmlichen Spiralknetern etwa drei Minuten lang mit verringerter Geschwindigkeit drehen. In dieser Zeit sinkt die gemessene Temperatur aufgrund der

Vermengung verhältnismäßig warmer Feststoffe mit kaltem Wasser. Dieser Verlauf kann durch die Gleichungen 4-9 nicht wiedergegeben werden.

Bild 4-17 zeigt einen Vergleich zwischen Simulationsergebnissen und Messungen. Man kann erkennen, dass das simulierte Motormoment prinzipiell dem gemessenen bzw. dem offline gefilterten Drehmoment gut folgt, dass es jedoch nur einen Teil der Rauschprozesse erklären kann. Unabhängig davon kann jedoch die Teigtemperatur in der Knetphase sehr genau wiedergegeben werden. Hierzu wurden die nötigen Werte für C_p und h_d u.a. identifiziert (siehe Bild 4-17 rechts, rote Kurve) und dabei das in [Kru] entwickelte Identifikationswerkzeug verwendet. Der grüne Verlauf entspricht einer Parametrierung des Modells mit den Literaturparametern aus [SCD+12].

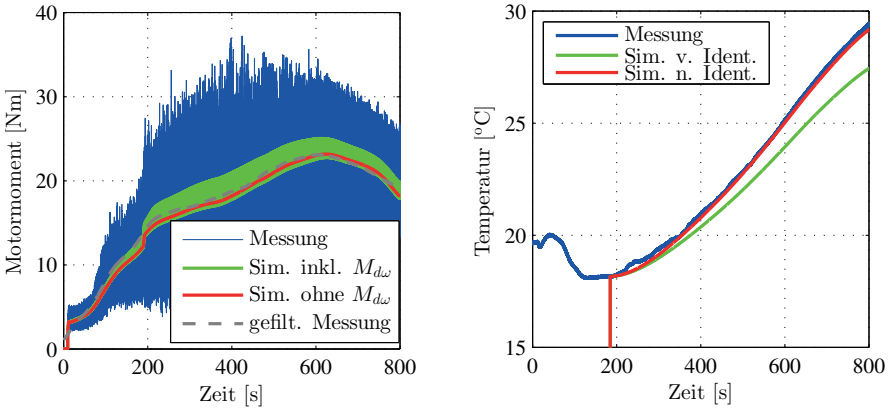


Bild 4-17: Vergleich zwischen Messung und Simulation[10]

Die drei beschriebenen Teilmodelle für Teig können einzeln oder in verschiedenen Kombinationen wiederverwendet werden, da sie durch den Bezug auf die elementare Größe „spezifische Energie“ soweit wie möglich unabhängig vom speziellen Einsatzzweck formuliert sind. Hierdurch können sie die Durchführung aufwendiger Messreihen teilweise ersetzen. Die identifizierten Parameter und Kennfelder sind zwar nur für die verwendete Teigrezeptur gültig, die mathematischen Zusammenhänge können jedoch unabhängig davon bestehen bleiben. Bei der Aufbereitung gilt es, dies entsprechend zu dokumentieren. Dazu zählen u. a. auch die getroffenen

[10] Allein durch diese Abbildung kann streng genommen nicht auf ein validiertes Temperaturmodell geschlossen werden. Hierzu ist der Vergleich mit einer Messung nötig, die nicht zur Parameteridentifikation genutzt wurde. An dieser Stelle sollen primär jedoch die Notwendigkeit und der Effekt der Identifikation gezeigt werden.

Annahmen sowie die beschränkte Gültigkeit des Temperaturmodells. Insbesondere am Beispiel des Letzteren wird der Unterschied zwischen den Größen Modellkomplexität und Modellierungstiefe deutlich. Die Modellierungstiefe liegt im Bereich von $\Gamma = 3$ und ist damit relativ hoch, da das Modell in seinem Gültigkeitsbereich die Realität sehr genau wiedergibt (vgl. Bild 4-17, rechts). Die Komplexität ist mit gerade einmal zwei Zuständen hingegen vergleichsweise klein. Es ergibt sich $K_{Teigtemperatur} = 2$.

5 Semantische Aufbereitung und Nutzung von Lösungswissen

Das Angebot konkreter Lösungselemente, sowohl von Zulieferern als auch im eigenen Unternehmen, ist in vielen Bereichen sehr groß. Dennoch gestaltet sich die Suche nach dem Lösungselement, das am besten geeignet ist, um eine Funktion zu erfüllen selbst dann schwierig, wenn die Anforderungen an das Lösungselement zum Beispiel durch eine idealisierte modellbasierte Analyse genau bekannt sind. Folgende Gründe können hierfür genannt werden.

(i) Die dem Entwickler zugänglichen Informationsquellen wie das Internet oder Produktkataloge basieren auf vielen unterschiedlichen, zum Teil herstellerspezifischen Klassifikationen, Terminologien und Darstellungen, die nicht ohne Weiteres vergleichbar sind. Die manuelle Suche nach dem optimalen Lösungselement ist daher aufwendig und in der Praxis vielfach nicht wirtschaftlich. Rechner können aktuell nur eine syntaktische Suche nach Lösungselementen leisten, denn sie verstehen die Semantik nicht, die hinter der Zeichenfolge steht. Eine semantische Suche, die alle Aspekte der Lösungselement-Spezifikation berücksichtigt, muss daher vom Entwickler selbst durchgeführt werden.

(ii) Für die Auswahl werden Kennwerte und Parameter angegeben. Das volle Potential unbekannter Lösungselemente kann auf dieser Basis jedoch nur schwer abgeschätzt werden. Gerade das reale dynamische Verhalten des Lösungselements lässt sich üblicherweise nicht nur durch statische Kennwerte beurteilen. Hierzu braucht es detaillierte Dynamikmodelle, welche darüber hinaus zur optimalen Auslegung der Informationsverarbeitung und zur modellbasierten Inbetriebnahme benötigt werden. Die Erstellung solcher Modelle ist aufwendig und an detailliertes Expertenwissen über das jeweilige Lösungselement geknüpft, welches beim Entwicklungsunternehmen im Unterschied zum LE-Hersteller normalerweise nicht vorhanden ist.

Beides führt dazu, dass Entwickler häufig nur bekannte und bewährte Lösungselemente auswählen, die zur Erfüllung der Aufgabe nicht in jedem Fall optimal geeignet sind, bzw. hohe Sicherheitsfaktoren einrechnen, um teure nachträgliche Anpassungen zu vermeiden (vgl. Kapitel 1). Das Potential des Produkts kann folglich nicht im vollen Umfang ausgeschöpft werden.

Bild 5-1 stellt eine Übersicht des Konzepts der effektiven Integration von Lösungswissen in den Entwurf intelligenter technischer Systeme mithilfe semantischer Technologien dar. In den jeweiligen Synthesephasen der Systemkonzipierung und

des disziplinspezifischen Entwurfs erfolgt die Suche nach wiederverwendbarem Lösungswissen (vgl. Kapitel 3). Dies geschieht durch Abfrage von Wissen aus einem Wissensmodell. Letzteres wird gewonnen, indem semantisch aufbereitetes Lösungswissen aus dem Semantic Web gesammelt („gecrawlt“) und mittels Inferenzmachinen (Reasonern) weiterverarbeitet wird (vgl. Abschnitt 2.4.4 auf Seite 43). Das konkrete Wissen über Lösungselemente wird von Zulieferern (Unternehmen B/C) zur Verfügung gestellt, die sich hierdurch neue Vertriebskanäle erschließen und auf ihr Angebot aufmerksam machen. Auch das eigene Unternehmen (Unternehmen A) stellt nach erfolgreich abgeschlossener Entwicklung das fertige Produkt als neues Lösungselement für andere Entwicklungen zur Verfügung. Allerdings wird es in der Regel im Sinne des Unternehmens sein, dass eigenes Lösungswissen teilweise nur innerbetrieblich genutzt wird. In diesem Fall ändert sich die semantische Aufbereitung nicht, stattdessen wird das entsprechende Wissen jedoch ausschließlich bei Anfragen aus dem internen Netz genutzt und nicht im globalen Semantic Web zur Verfügung gestellt. Übergeordnetes Wissen über die Zusammenhänge zwischen Lösungselementen und deren Zugehörigkeit zu Lösungsmustern sollte durch eine unabhängige Standardisierungsorganisation bzw. ein Konsortium eingepflegt und überwacht werden. Dadurch wird Chancengleichheit hergestellt und Vertrauen geschaffen. Auch die Bereitstellung der Lösungsmuster selbst (vgl. Bild 4-1 auf Seite 86) sowie der idealisierten Lösungsmustermodelle sollte von dieser durchgeführt/beaufsichtigt werden. Da Lösungsmuster eine abstrakte Darstellung einer Klasse von Lösungselementen darstellen [GTS14, S. 123], ist die Klassifizierung dieser Voraussetzung für die Definition von Lösungsmustern. Letztere basiert auf der Identifikation verbindender Eigenschaften. Anschließend werden semantische Beziehungen zwischen dem einzelnen Lösungselement mit seinen Aspekten und dem zugehörigen Lösungsmuster identifiziert und modelliert.

Im Folgenden wird fokussiert, wie Lösungselemente mithilfe semantischer Technologien aufbereitet und gefunden werden können. Dies erfolgt am Beispiel von Aktoren. Die Arbeiten wurden im Rahmen des Forschungsprojekts ENTIME durchgeführt. Teile davon finden sich daher bereits in [GTS14]. Die semantische Suche nach Lösungsmustern kann auf vergleichbare Art und Weise erfolgen, jedoch mithilfe von Anforderungen und Funktionen. Sie ist ebenfalls in [GTS14] beschrieben und wird hier ausgeklammert.

5.1 Semantische Aufbereitung von Lösungswissen

Kapitel 3 beschreibt die Nutzung von Lösungswissen im Entwurf mechatronischer Systeme. Das große Potential semantischer Technologien für die Integration des Wissens wurde ebenfalls bereits herausgearbeitet. Um dieses nutzen zu können, muss es jedoch zunächst entsprechend aufbereitet werden. Im Folgenden wird die Spezifikation und semantische Repräsentation von Lösungselementen beschrieben.

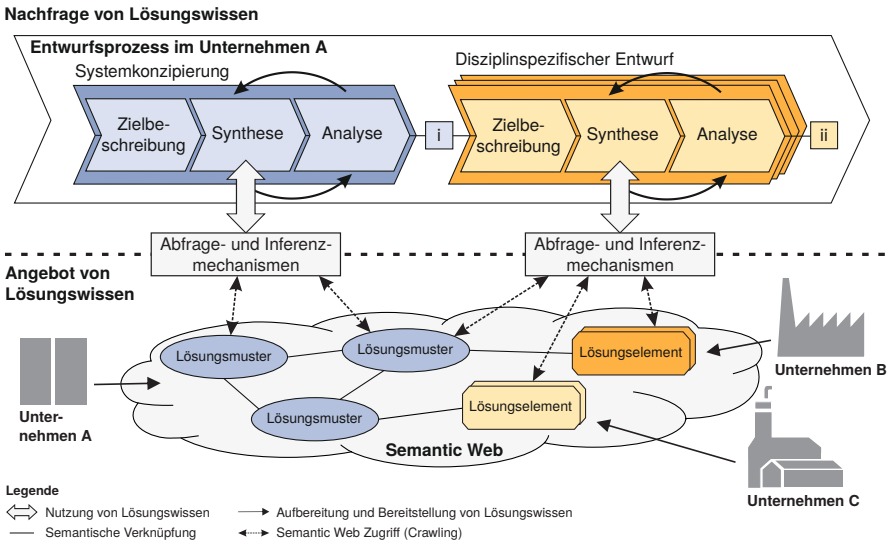


Bild 5-1: Konzept der Integration von semantisch aufbereitetem Lösungswissen durch Angebot und Nachfrage, vgl. [GSA+11, OJT+12]

5.1.1 Spezifikation von Lösungselementen

Bevor Lösungselemente semantisch aufbereitet werden können, ist zunächst herauszuarbeiten, welche Informationen Anbieter – unabhängig davon, ob sie aus dem eigenen oder aus Zulieferunternehmen kommen – für diese bereitstellen müssen. Ziel ist es, dem Entwickler bzw. dem Nachfrager von Lösungswissen möglichst alle benötigten Daten zu liefern, sodass dieses nahtlos in seinen Entwurfsprozess integriert werden kann. Vor dem Hintergrund des aufgezeigten Entwurfsvorgehensmodells (vgl. Kapitel 3) wurde eine Spezifikation erarbeitet, die in Bild 5-2 am Beispiel des physikalischen Servomotors *AM8023* der Firma Beckhoff dargestellt ist. Sie beinhaltet folgende sechs Aspekte, welche zur Bereitstellung eines Lösungselements beschrieben werden müssen, damit sie zur Verringerung der Komplexität und Fehleranfälligkeit des Entwurfsprozesses beitragen können [GTS14]:

Auswahlmerkmale: Nenngrößen und Kennwerte ermöglichen es, die Anzahl der Lösungselemente einzuschränken. Neben diesen Zahlenwerten geben Merkmale wie *Produktbezeichnung*, *Hersteller* oder *Schutzklasse* textuell Auskunft.

Kontext: Der Aspekt Kontext beinhaltet aussagekräftige Beispielanwendungen, in denen das Lösungselement verwendet wurde. Die Beschreibung erfolgt auf textueller bzw. grafischer Basis. Außerdem werden Produktdokumentationen bereitgestellt.

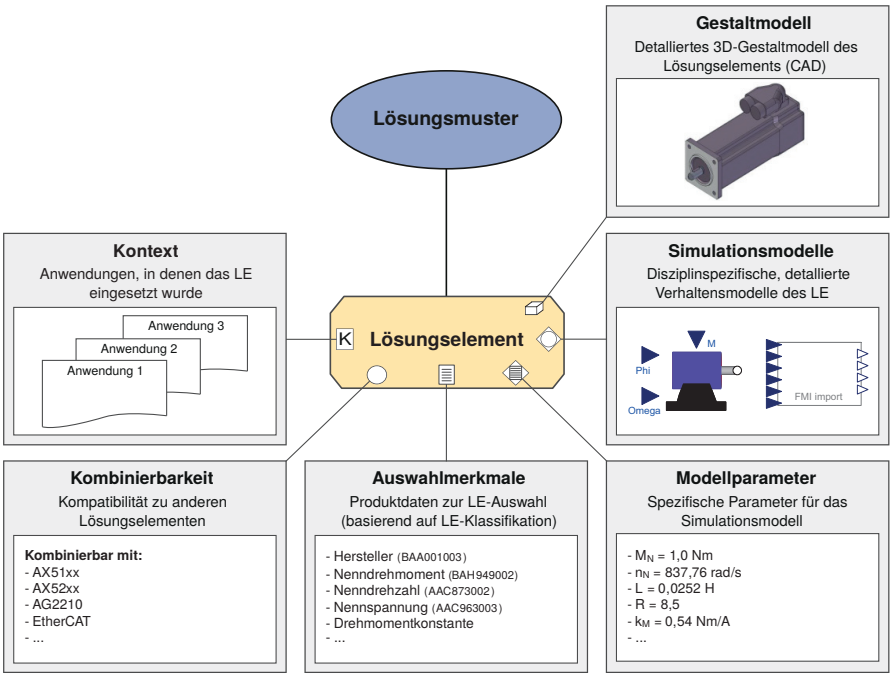


Bild 5-2: Einheitliche Spezifikation von physikalischen Lösungselementen [GTS14, S. 102]

Gestaltmodell: Die Bereitstellung eines 3D-Gestaltmodells in gängigen Austauschformaten wie STEP ist heute ein „Quasi-Standard“. Ein solches Modell kann für die konstruktive Ausgestaltung im Rahmen der Ausarbeitung des Systems genutzt werden. Handelt es sich um ein immaterielles Lösungselement, entfällt dieser Aspekt.

Verhaltens-/Simulationsmodelle: Detaillierte Verhaltensmodelle erlauben die Optimierung und Analyse unter realistischen Bedingungen. Im Zuge der disziplinspezifischen Ausarbeitung sowie der disziplinübergreifenden Koordination kommen hierfür z. B. Dynamikmodelle zum Einsatz. Es ist erforderlich, dass die Modelle das produktspezifische, charakteristische dynamische Verhalten detailliert wiedergeben. Es werden Dynamikmodelle gemäß Abschnitt 4.4.1 aufbereitet und bereitgestellt (z. B im FMI-Austauschformat). Für ähnliche Lösungselemente einer Klasse kann vielfach dasselbe LE-Modell verwendet werden. Software-Lösungselemente können darüber hinaus durch andere Verhaltensmodelle (zum Beispiel durch UML-Diagramme) beschrieben werden.

Modellparameter: Um das zur Verfügung gestellte Modell einer Lösungselementklasse an das spezifische Lösungselement anzupassen, bedarf es der Bereitstellung der hierfür nötigen Parameter. Diese stehen vielfach in engem Zusammenhang mit den Auswahlmerkmalen.

Kombinierbarkeit: Die Kombinierbarkeit bzw. Kompatibilität mit anderen, verwendeten Lösungselementen desselben oder anderer Hersteller stellt einen essenziellen Teilaspekt bei der Auswahl dar. Mithilfe einer Positivliste sollen diejenigen Lösungselemente angegeben werden, deren Funktionsfähigkeit in Kombination mit diesem Lösungselement nachgewiesen wurde.

Um die korrekte Eingliederung der aufbereiteten Lösungselemente zu erleichtern, bietet sich die Nutzung bestehender Produktklassifikationen als Basis an. Diese können in dem allgemeingültigen, terminologischen Teil des Wissensmodells (T-Box) als eine Sicht auf die Daten hinterlegt werden (vgl. Abschnitt 2.4.1 auf Seite 30). Klassifiziert ein Anbieter seine Produkte gemäß des Standards, so ist dadurch die grundlegende Verknüpfung und korrekte Eingliederung in die durch die T-Box gegebene Terminologie implizit gegeben. Der Standard eCl@ss wurde als geeignete Grundlage für die Klassifikation von Lösungselementen identifiziert, da er sowohl international als auch branchenübergreifend angelegt und zudem mit anderen wichtigen Produktklassifikationsstandards harmonisiert ist. Er bietet bereits eine Vielzahl von Klassen und Merkmalen zur Beschreibung, die jedoch aufgrund der Ausrichtung auf den Handel nach technischen Gesichtspunkten nicht immer ausreichend sind. An diesen Stellen werden zusätzliche Merkmale eingeführt. Die Tabelle 5-1 zeigt die für das Beispiel des Beckhoff-Servomotors AM8023E genutzten Lösungselement-Merkmale im Detail. Sie orientieren sich an den Merkmalen des Standards eCl@ss (Version 6.2) [ECL09]. In diesem Fall werden sechs weitere Merkmale hinzugefügt.

5.1.2 Semantische Repräsentation

Nachfolgend werden vornehmlich die T-Box-Anteile der Wissensbasis erläutert, die später dazu genutzt werden können, konkretes Lösungswissen zu beschreiben und so für die Wissensbasis aufzubereiten. Sie stellen Ausdrucksmittel im Sinne einer sprachbasierten Metaisierung dar (vgl. Abschnitt 2.1.3). Die Nutzung dieser Ausdrucksmittel zur Befüllung der A-Box, d. h. zur Modellierung konkreten Wissens, wird in Abschnitt 5.2 beispielhaft gezeigt. Beides ist Voraussetzung, damit der Entwickler die semantische Suche nutzen kann, und stellt gewissermaßen die technische Ausgestaltung der in Bild 5-1 als Wolke illustrierten Wissensbasis dar. Diese gilt es anschließend mit einer entsprechenden Benutzerschnittstelle und Erklärungskomponente auszustatten (vgl. Bild 2-11 auf Seite 31), damit die Suche sich für den Nutzer nahtlos in den Entwicklungsprozess eingliedert und keine spezifischen Kenntnisse oder Schulungen erfordert (vgl. Abschnitt 5.2.3).

Tabelle 5-1: Auswahlmerkmale des Beckhoff-Servomotors AM8023E [GTS14, S. 104]

Auswahlmerkmal	eCl@ss-Merkmal	Wert
Herstellername	BAA001003	Beckhoff Automation
Produktbezeichnung	BAA316003	AM8023-wEyz
Nennspannung	AAC963003	400 V
Nennleistung	AAC967002	0,84 kW
Nenndrehmoment	BAH949002	1,0 N m
Nenndrehzahl	AAC873002	8000 rpm
Nennstrom	AAC134002	1,85 A
Spitzendrehmoment	BAE098003	6 N m
Maximalstrom	AAC130003	11,4 A
Maximale Drehzahl	BAA120004	12 000 rpm
Rotorträgheit	AAC142003	0,373 kg cm^2
Kühlung	BAE122002	Konvektion
Schutzklasse	BAG342002	IP 54, IP 65
Gewicht	BAD875004	1,7 kg
Höhe	BAA020004	82,4 mm
Breite	BAF016003	63 mm
Tiefe	BAB577004	175,5 mm
Flansch	BAE096001	58 mm
Thermische Zeitkonstante	-	16 min
Reibmoment	-	0,006 N m
Wicklungsinduktivität	-	25,2 mH
Wicklungswiderstand	-	8,5 Ω
Drehmomentkonstante	-	0,54 N m/A
Spannungskonstante	-	43 mV min

Für die semantische Modellierung der spezifizierten Aspekte muss zunächst das nötigen Wissens identifiziert werden, dessen Semantik im Anschluss in geeigneter Form zu modellieren ist. Hierbei muss – wie bei jedem Modell (vgl. Abschnitt 2.1.1) – der Zweck im Vordergrund stehen. Im Mittelpunkt soll die Suche nach Lösungswissen stehen, es ist daher zum Beispiel wenig sinnvoll den inneren Aufbau des Simulationsmodells semantisch aufzubereiten und für dieses ein semantisches Metamodell zu erstellen. Vielmehr geht es darum, diejenigen Informationen abzubilden, die zum Auffinden des geeigneten Lösungselements und des passenden Simulationsmodells nötig sind. Das Simulationsmodell selbst wird auf einem entsprechend zugänglichen Daten-Server (Repository) abgelegt. Im ersten Schritt werden On-

tologien benötigt, welche die nötigen terminologischen Konstrukte beinhalten. In diesem Zusammenhang sind Informationen über die Auswahlmerkmale, die Kombinierbarkeit mit anderen LE, den Autor, die verwendeten Modellsprachen und wesentliche Parameter von Interesse. Darüber hinaus hilft die Modellierung allgemeingültiger, logischer Zusammenhänge später dabei, implizit vorhandenes Wissen aus den explizit modellierten Informationen herzuleiten.

Bei den Konstrukten zur expliziten Modellierung der T-Box handelt es sich im Wesentlichen um Definitionen von Klassen, Objektmerkmalen und Datenmerkmalen. In der A-Box werden vornehmlich konkrete Individuen dieser Klassen beschrieben. Beides geschieht mit der Ontologiesprache OWL-DL. Um die Wiederverwendbarkeit auch in anderen Anwendungsbereichen zu erhöhen, werden die Ontologien nach Themenbereichen aufgeteilt und benannt. Darüber hinaus bleibt hierdurch nach dem Prinzip *teile und herrsche* der Umfang der einzelnen Teile besser beherrschbar. Zwar wird die Wartung aufgrund der verschiedenen Querverbindungen teilweise erschwert, jedoch beinhaltet gerade die aufwendig zu wartende T-Box grundlegende Zusammenhänge, die sich normalerweise nicht ändern. Tabelle 5-2 zeigt die Aufteilung sowie verwendete Präfixe und Namensräume. Die letzte Spalte ordnet den einzelnen Präfixen jeweils eine Farbe zu, welche in nachfolgenden Diagrammen als Wiedererkennungsmerkmal genutzt wird. Konkreteres Wissen korrespondiert dabei mit intensiverer Färbung.

Konstrukte zur Modellierung von explizitem Lösungswissen

Konstrukte zur Modellierung von explizitem Lösungswissen sind Ausdrucksmittel, d. h. Klassen und Merkmale, die dazu dienen, konkretes Wissen unmittelbar zu beschreiben. Als Beispiel sind die Tripel (Subjekt-Prädikat-Objekt-Verbindungen) zur Definition der übergeordneten Klasse `SolutionElement` in Quellcode 5-1 zu sehen (vgl. Abschnitt 2.4.3 auf Seite 33). Um eine für den Menschen verständlichere Beschreibung einzuführen, wird dieser neben der URI je ein deutsch- und ein englischsprachiges Label zugewiesen. Weiterhin werden mithilfe sogenannter *Restrictions* Bedingungen/Eigenschaften eingeführt, die alle Individuen dieser Klasse erfüllen sollen (Zeilen 5–20). Sie sollen über das Objektmerkmal `hasSEModel` mit einem detaillierten Lösungselement-Modell (`SolutionElementModel`) verknüpft sein. Durch das Merkmal `hasSolutionPattern` besteht die Verbindung zu Lösungsmustern (`SolutionPattern`) und mittels `hasSEProducer` zu einem Hersteller. Zuletzt ist auch die Angabe von Auswahlmerkmalen (`SolutionElement-Characteristic`) erforderlich, um das passende Lösungselement herausfiltern zu können. Die Kombinierbarkeit kann durch das Objektmerkmal `isCombineable` ausgedrückt werden, dessen Tripel in den Zeilen 22–24 dargestellt sind.

Tabelle 5-2: Verwendete Abkürzungen für Namensräume, vgl. [GTS14, S. 101]

Präfix	Erläuterung	Farbe
entime:	Beschreibung übergeordneter und allgemeingültiger Zusammenhänge zwischen Lösungselementen, Lösungsmustern, Funktionen, Anforderungen und Modellen (T-Box) *http://entime.uni-paderborn.de/ontology/ENTIME#*	
cd:	T-Box-Ontologie, die Konstrukte der Systemkonzipierung (engl. **C**onceptional**D**esign) definiert, wie z. B. Lösungsmuster, Funktionen und Anforderungen *http://entime.uni-paderborn.de/ontology/ConceptualDesign#*	
spa:	Beinhaltet das Wissen über die konkreten Lösungsmuster-Individuen der A-Box (**S**olution**P**attern**A**Box) *http://entime.uni-paderborn.de/SolutionPatterns/SolutionPatternABox#*	
eclass:	Nach eCl@ss 6.2 standardisierte Klassen für Lösungselemente *http://entime.uni-paderborn.de/ontology/eClass6.2#*	
se:	Terminologien (T-Box) zur Beschreibung eines Lösungselements (**S**olution**E**lement) *http://entime.uni-paderborn.de/ontology/SolutionElement#*	
sea:	Konkretes, herstellerübergreifendes Lösungswissen über Lösungselemente (**S**olution**E**lement**A**Box) *http://entime.uni-paderborn.de/SolutionElements/SolutionElementABox#*	
model:	T-Box-Elemente zur Beschreibung von Modellen im Entwurfsprozess, z.B. Simulationsmodelle für Lösungselemente und Lösungsmuster *http://entime.uni-paderborn.de/ontology/EngineeringModel#*	
modelLib:	Umfasst die relevanten Informationen über konkret vorhandene Lösungsmuster-, Interface- und herstellerübergreifende Lösungselementmodelle (**ModelLib**raryABox) *http://entime.uni-paderborn.de/SimulationModels/SimulationModelLibrary#*	

Quellcode 5-1: Ontologieausschnitt zur Definition der übergeordneten Lösungselement-Klasse in Turtle-Syntax

```
1  se:SolutionElement rdf:type owl:Class;
2          rdfs:label "Lösungselement"@de;
3          rdfs:label "Solution Element"@en;
4          rdfs:subClassOf
5             [ rdf:type owl:Restriction;
6               owl:onProperty se:hasSEModel;
7               owl:allValuesFrom model:SolutionElementModel
8             ] ,
9             [ rdf:type owl:Restriction;
10              owl:onProperty se:hasSolutionPattern;
11              owl:allValuesFrom cd:SolutionPattern
```

```
12            ] ,
13            [ rdf:type owl:Restriction;
14              owl:onProperty se:hasSEProducer;
15              owl:allValuesFrom se:SolutionElementProducer
16            ] ,
17            [ rdf:type owl:Restriction;
18              owl:onProperty se:hasSolutionElementCharacteristic;
19              owl:allValuesFrom se:SolutionElementCharacteristic
20            ].
21
22 se:isCombineable rdf:type owl:ObjectProperty;
23                  rdfs:domain se:SolutionElement;
24                  rdfs:range se:SolutionElement.
```

Im Folgenden wird zusätzlich zur textuellen eine grafische Darstellung zur Erläuterung der verwendeten Klassen, Merkmalen und Individuen gewählt. Diese orientiert sich an der UML/SysML, nutzt Analogien zu UML-Klassendiagrammen aus (vgl. Abschnitt 2.4.3 auf Seite 33) und ermöglicht einen zwar immer noch nicht umfassenden, aber größeren Überblick über die modellierten Zusammenhänge. Bild 5-3 zeigt einen Ausschnitt aus der T-Box des Wissensmodells (vgl. Abschnitt 2.4.1 auf Seite 30).

Das Diagramm beinhaltet verschiedene Klassen zur Beschreibung von Lösungselementen, Lösungsmustern, Modellen (sowohl von Lösungsmustern als auch Lösungselementen) und deren Charakteristika. Dazu können Informationen wie Beschreibungen, genutzte Abkürzungen und auch die verwendete Einheit hinterlegt werden. Bild 5-3 stellt hierfür wichtige Verbindungen (Objektmerkmale) dar, die zwischen den Individuen der Klassen gezogen werden können. Wertbehaftete Merkmale (Datenmerkmale) sind auszugsweise als Attribute der Klassen veranschaulicht. Mithilfe dieser Konstrukte lässt sich zum Beispiel modellieren, dass ein Lösungselement andere Lösungselemente beinhaltet (`comprisesSolutionElement`) sowie genau einen Hersteller (`hasSolutionElementProducer`) und Kontexte (`hasSolutionElementContext`) haben kann. Es kann weiterhin neben harten, quantitativen Auswahlmerkmalen (`SolutionElementCharacteristic`) auch weiche (`SoftCharacteristic`) Charakteristika besitzen, die textuell beschrieben sind. Die Verbindung zu abstrakteren Lösungsmustern wird über die Kante `hasSolutionPattern` gezogen. Darüber hinaus wird die Klassifikation nach eCl@ss integriert und den durch diesen Standard spezifizierten Klassen die Klasse `eClass` übergeordnet. Zwischen Letzterer und der Klasse `SolutionElement` ist eine Generalisierungsbeziehung modelliert, sodass alle Individuen, die gemäß des Standards eCl@ss klassifiziert sind, als Lösungselement angesehen werden.

Es hat sich allerdings herausgestellt, dass eine direkte Zuordnung von Lösungsmustern zu den durch eCl@ss definierten Lösungselementklassen nicht immer möglich ist, da der Standard nicht nur nach technischen Kriterien aufgebaut ist

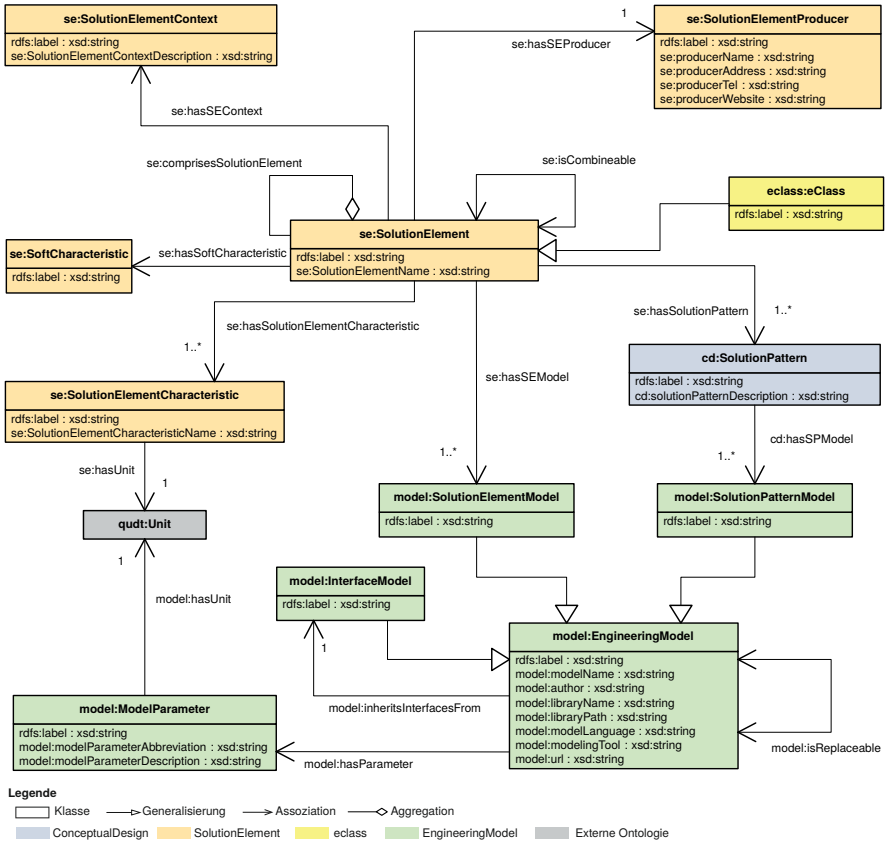


Bild 5-3: Ausschnitt der T-Box des Wissensmodells, dargestellt als UML-Klassendiagramm, vgl. [GTS14, S. 109]

und z. B. auch Dienstleistungen klassifiziert. Die Aufgabe des Wissensmodells ist es u. a. jedoch, eine solche Verknüpfung zwischen Lösungsmustern und Lösungselementen zu schaffen. Aus diesem Grund ist die Einführung von weiteren, übergeordneten Klassen notwendig, um eine gezielte Suche zu ermöglichen. Ein wesentlicher Vorteil von Ontologien im Vergleich zu reinen Datenbanken ist, dass sie die effiziente Organisation von vielen verschiedenen „Sichten“ auf das abgelegte Wissen erlauben. So stellt die Klassifizierung nach eCl@ss nur eine solche Sicht dar. Auf Grundlage intensiver Literaturrecherche (siehe u. a. [Ise08, Czi06]) wird darüber hinaus für den Bereich der Aktoren u. a. die in Bild 5-4 dargestellte Klassifikation integriert. Diese unterteilt nach der zugrundeliegenden Energieart z. B. in elektromechanische und fluidenergie Aktoren sowie unkonventionelle Aktuatoren, wie bspw. piezoelektrische Antriebe. Weiterhin werden die teilweise mehrdimensionalen, hierarchischen Beziehungen zu den eCl@ss-Klassen abgebildet. Da die dargestellte Klassifikation anhand des technischen Funktionsprinzips erfolgt ist, können den einzelnen Klassen Restriktionen (vgl. Quellcode 5-1) zugeordnet werden, die dafür sorgen, dass alle Individuen der Klasse verbindende, technische Eigenschaften erben. Dies ermöglicht die effiziente semantische Abspeicherung und Aufbereitung von Expertenwissen [ODB+12, OJT+12]. Beispielsweise zeichnen sich elektromechanische Aktoren generell durch eine gute Regelbarkeit aus. Quellcode 5-2 zeigt die Modellierung solchen Wissens anhand dieser Aussage. Neben der Klassendefinition für elektromechanische Aktoren inkl. der Restriktionen sind die Tripel für das weiche Charakteristikum `goodControllability` aufgeführt (Individuum).

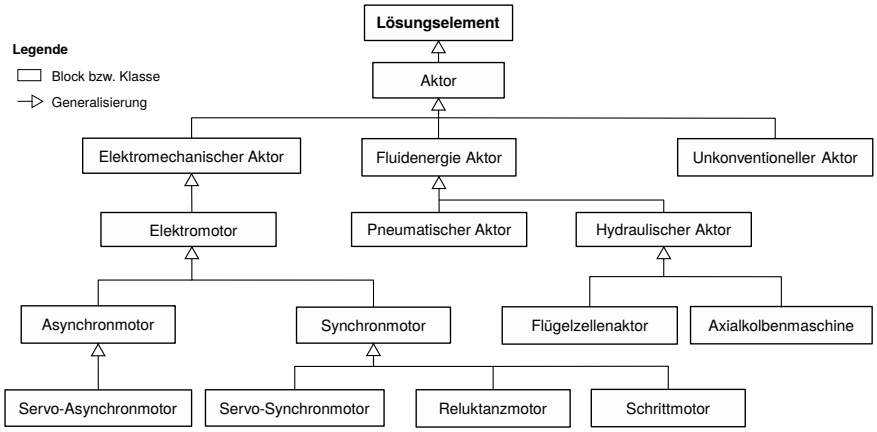


Bild 5-4: Ausschnitt der LE-Klassifikation für Aktoren vgl. [GTS14, S. 110]

Quellcode 5-2: Tripel zur Definition der Klasse „Elektromechanischer Aktor“ sowie weicher Auswahlmerkmale

```
sea:ElectromechanicalActuator rdf:type owl:Class;
        rdfs:label "Elektromechanischer Aktor"@de;
        rdfs:label "electromechanical actuator"@en;
        rdfs:subClassOf se:SolutionElement,
                        sea:Actuator,
                      [ rdf:type owl:Restriction;
                        owl:onProperty se:hasSoftCharacteristic;
                        owl:hasValue sea:goodControllability
                      ],
                       ...

sea:goodControllability rdf:type owl:NamedIndividual,
                        se:SoftCharacteristic;
                        rdfs:label "gute Regelbarkeit"@de,
                                   "good controllability"@en.
```

Die Zuordnung von herstellerübergreifenden, detaillierten Lösungselementmodellen (vgl. Abschnitt 4.4.1) kann auf analoge Art und Weise erfolgen. Zu diesem Zweck wird für jedes Simulationsmodell ein Individuum der Klasse `SolutionElementModel` erzeugt. Die Verknüpfung mit den zugehörigen Lösungselement-Individuen geschieht mittels des Objektmerkmals `hasSEModel`. Die zur Suche und Auswahl weiterhin nötigen Meta-Informationen werden mithilfe von Datenmerkmalen bereitgestellt. Dazu zählen z. B. die Bezeichnung (`modelName`), der Autor (`author`), die Modellierungssprache (`modelLanguage`), das Modellierungswerkzeug (`modelingTool`) und die URL, unter der die Modelldatei herunterzuladen ist (vgl. Bild 5-3). Modellparameter werden analog zu quantitativen LE-Merkmalen abgebildet, um auch diesen Meta-Informationen wie z. B. die Einheit, eine Beschreibung oder auch Informationen über eine erfolgte Parameteridentifikation „mitgeben“ zu können (vgl. Kapitel 6). Besteht ein Zusammenhang zwischen einem Lösungselementmerkmal und und einem Modellparameter, so wird dieser mithilfe des Objektmerkmals `correspondsTo` beschrieben (vgl. Abschnitt 5.2). Modellierungswerkzeuge können dies nutzen, um z. B. eine automatische Parametrierung von Lösungselement-Modellen zu realisieren.

Konstrukte zur Herleitung von Lösungswissen

Neben der Schaffung von expliziten Ausdrucksmitteln ermöglichen es Ontologien, Wissen so zu modellieren, dass logische Schlussfolgerungen gezogen werden können. Nach der Auswertung durch einen Reasoner (vgl. Abschnitt 2.4.4 auf Seite 43) kann dieses implizit vorhandene Wissen auf gleiche Art und Weise abgefragt werden. Inferenzen können sich zum Beispiel bereits aus der zugrundeliegenden Klassifikation der Individuen ergeben. Es können Schnittmengen, Äquivalenzen

und disjunkte Klassen modelliert werden. Zum Beispiel kann dadurch, dass der erwähnte Servo-Synchronmotoren `AM8023E` zur Klasse der permanenterregten Motoren gehört, für ihn ein zu erwartender Wirkungsgradvorteil geschlussfolgert werden. Denn dieser Wirkungsgradvorteil besteht für alle permanenterregten Motoren gegenüber fremderregten Maschinen, sodass das entsprechende, weiche Kriterium der übergeordneten Klasse geerbt wird. Er resultiert aus dem Wegfall von Magnetisierungsenergie und gilt u. a. auch für bürstenlose Gleichstrommotoren, die ebenfalls zur Klasse der permanenterregten Motoren gehören. Gemeinsame Charakteristika aller Synchronmotoren sind zum Beispiel, dass sie vergleichsweise kostengünstig und wartungsfrei sind, jedoch ggf. Probleme beim Anlauf bereiten [Ise08].

Eine andere Möglichkeit zur Erzeugung von Inferenzen ist die Nutzung von erweiterten Merkmalen/Verknüpfungen. Schon die Definition von *domain* und *range* – also des Start- und Endknotens einer Kante – ermöglicht u. U. Schlussfolgerungen über die Zugehörigkeit eines Individuums. Wird beispielsweise die Beziehung `hasSEModel` zwischen einem Individuum, das ansonsten bisher noch keiner Klasse zugeordnet wurde, und einem LE-Modell gezogen, so kann aufgrund der Information über den Startknoten der Kante geschlussfolgert werden, dass es sich bei dem Individuum um ein Lösungselement handeln muss. Die Tripel zur Definition von Start- und Endknoten von `hasSEModel` sind in den Zeilen 2 und 3 des Quellcodes 5-3 gezeigt. Darüber hinaus werden Merkmalhierarchien aus über- und untergeordneten Merkmalen verwendet. Die spezielle Beziehung `hasSEModel` wird z. B. dem abstrakteren Merkmal `hasModel` untergeordnet (vgl. Zeile 4). Letzteres gilt dadurch immer implizit, wenn Verknüpfungen zwischen Individuen und Lösungselement-, Lösungsmuster- bzw. Interface-Modellen verwendet werden. Abstrakte Anfragen nach `hasModel` liefern folglich alle entsprechenden Individuen zurück, die über eines der vier Merkmale verknüpft sind.

Quellcode 5-3: Tripel zur Definition der Verknüpfung von Lösungselementen mit LE-Modellen

```
1 se:hasSEModel rdf:type owl:ObjectProperty;
2               rdfs:domain se:SolutionElement;
3               rdfs:range model:EngineeringModel;
4               rdfs:subPropertyOf model:hasModel;
5               owl:inverseOf model:isSEModelOf.
```

Mithilfe von OWL-Sprachbausteinen werden symmetrische, transitive, funktionale, inverse und sich ausschließende (disjoint) Merkmale definiert (vgl. Tabelle 2-3 auf Seite 42). Durch das Tripel in Zeile 5 des Quellcodes 5-3 wird so beispielsweise erreicht, dass jeweils die entgegen `hasSEModel` gerichtete Kante `isSEModelOf` inferiert/hergeleitet wird und andersherum. Das jeweilige Wissen ist deshalb auch

dann verfügbar, wenn „von der anderen Seite aus“ gefragt wird. Ein weiteres Beispiel ist die Beziehung `isReplaceable`. Diese wird zwischen Modellen gezogen, um auszudrücken, dass sie gegeneinander ausgetauscht werden können. Sie ist symmetrisch, also immer in beide Richtungen navigierbar. Gleichzeitig ist sie transitiv, denn es gilt: Wenn jeweils Modell A und B sowie Modell B und C austauschbar sind, dann besteht die Verknüpfung auch zwischen A und C.

Für komplexere logische Beziehungen werden global und einmalig definierte SWRL-Regeln verwendet. Ein erstes Beispiel zur Herleitung von Kombinierbarkeitsbeziehungen wurde bereits zur Erläuterung von SWRL in Abschnitt 2.4.3 auf Seite 41 gezeigt. Die nachfolgende Regel wird genutzt, um die erwähnte Austauschbarkeitsbeziehung zwischen Modellen herleiten zu können. Die Basis hierfür bilden die Interface-Modelle (vgl. Bild 4-5 auf Seite 96). Durch die Regel wird erreicht, dass zwischen Modell-Individuen, die über die Verknüpfung `inheritsInterfacesFrom` mit demselben Interface-Modell verbunden sind, das Merkmal `isReplaceable` inferiert wird.

$$\begin{aligned} &\mathit{EngineeringModel(?model1)} \wedge \mathit{EngineeringModel(?model2)} \wedge \\ &\mathit{InterfaceModel(?interfaceModel)} \wedge \\ &\mathit{inheritsInterfacesFrom(?model1,?interfaceModel)} \wedge \\ &\mathit{inheritsInterfacesFrom(?model2,?interfaceModel)} \\ &\rightarrow \mathit{isReplaceable(?model1,?model2)} \end{aligned}$$

Auf ähnliche Art und Weise wird eine Vorschrift formuliert, sodass konkreten Lösungselementen auch die idealisierten Modelle der zugehörigen Lösungsmuster zugeordnet werden[1]. Weiterhin wird mithilfe von Regeln eine automatische Verknüpfung von Modellen bzw. Lösungselementen mit Charakteristika erreicht, die mit bereits bestehenden Auswahlmerkmalen/Parametern korrespondieren (`correspondsTo`). Für die Suche nach Lösungselementen kommt vor allem dieser Verknüpfung besondere Bedeutung zu, denn der Suchraum wird insbesondere durch quantitative Merkmale verkleinert. Modellparameter als ein Ergebnis der modellbasierten Analyse sollen als Auswahlkriterien für Lösungselemente fungieren (vgl. Abschnitt 5.2). Jedoch besteht gerade zwischen Modellparametern und Lösungselement-/Lösungsmustermerkmalen aus vielerlei Gründen oftmals keine direkte Äquivalenz. An dieser Stelle wird die Beziehung `correspondsTo` verwendet.

SWRL bietet auch die Möglichkeit, Umrechnungsvorschriften für Datenmerkmale zu hinterlegen, beispielsweise kann bei Getriebeantrieben das Nennmoment mithilfe des Übersetzungsfaktors von der Antriebs- auf die Abtriebsseite umgerechnet

[1] Eine Alternative für diesen speziellen Anwendungsfall wäre auch eine sog. *property chain* (Merkmalskette) [HKP+12]. Diese sind jedoch weniger ausdrucksstark. Um eine möglichst einfache Wartbarkeit der Regelkonstrukte zu gewährleisten, wurden sie ausgeklammert.

werden. So ließen sich auch Einheiten umrechnen (vgl. [OJT+12]), jedoch muss dies für jede Merkmalkombination und Rechenrichtung einzeln erfolgen. Mit Blick auf eine flächendeckende Nutzung erscheint es daher praktikabler, lediglich die Information über die Einheit sowie den Umrechnungsfaktor zur jeweiligen SI-Einheit verfügbar zu machen (vgl. Bild 5-3). Dies geschieht durch die Nutzung der sog. *Quantities, Units, Dimensions and Data Types Ontologies (QUDT)*[2] der NASA [HKH+14]. Die eigentliche Umrechnung erfolgt stattdessen vor bzw. nach der eigentlichen Anfrage im jeweiligen Werkzeug.

Meta-Modellierung mittels Punning

Es fällt auf, dass Lösungselementmerkmale jeweils sowohl als Individuum als auch als Datenmerkmal existieren. Der Grund für diese Art der Modellierung, die sich *Punning* nennt, soll im Folgenden erläutert werden.

Neben der Abbildung des konkreten Werts ist es das Ziel, Meta-Informationen über Lösungselementcharakteristika zu beschreiben. Hierzu bietet sich die Modellierung von LE-Merkmalen als Individuen an. Meta-Informationen wie synonyme Bezeichnungen, der Geltungsbereich, Minimal- und Maximalwerte etc. können dann mithilfe von Datenmerkmalen für das jeweilige Individuum modelliert werden. Weiterhin ist es sehr einfach möglich, Verknüpfungen zwischen zwei Merkmalen oder die Verknüpfung mit QUDT-Einheiten (siehe oben) über Objektmerkmale darzustellen. Will man jedoch ausschließlich Individuen nutzen, so bedeutet das, dass wenn ein Lösungselementmerkmal einen konkreten Wert besitzt, für jeden unterschiedlichen Wert ein neues Individuum erstellt werden muss. Bei der großen Menge an Lösungselementen mit sehr vielen verschiedenen Werten, zum Beispiel für das Nennmoment, erscheint dies nicht mehr praktikabel. Zwar können Individuen anonym als sog. *blank nodes* formuliert werden, die keine explizite URI besitzen, allerdings erhöhen diese die Komplexität – insbesondere bei der Wartung der Wissensbasis – und verursachen auch darüber hinaus u. U. Probleme (siehe z. B. [MAH+11]). Einige Autoren wie Heath und Bizer raten daher sogar komplett von der Nutzung von *blank nodes* ab [HB11].

Zur Beschreibung von konkreten Werten für Lösungselementmerkmale sind Datenmerkmale (data properties) deutlich besser geeignet. Eine direkte Verbindung zwischen einem Datenmerkmal und einem Individuum mittels eines Objektmerkmals ist in der OWL-Sprachversion DL (description logic) jedoch nicht möglich. Hierdurch käme man in den Bereich von *OWL Full*, was bedeuten würde, dass die Ontologie nicht mehr komplett logisch eindeutig („entscheidbar“) wäre und

[2] Die QUDT-Ontologien stellen Ausdrucksmittel für Mengen (Quantities), Einheiten (Units), Dimensionen (Dimensions) und Datentypen (Data Types) zu Verfügung. Sie beinhalten bspw. sämtliche Einheiten des internationalen Einheitensystems (SI-Einheiten).

demzufolge von keinem Reasoner mehr vollständig unterstützt würde. Um beide Vorteile nutzen zu können, wird sogenanntes *Punning*, d. h. eine kontextabhängige Semantik (engl. contextual semantics), verwendet [Mot07, HKP+12]. Diese einfache Art der Meta-Modellierung wurde in der *OWL 2 DL* eingeführt. Demnach können verschiedene Entitäten unterschiedlichen Typs die gleiche Bezeichnung (URI) besitzen. Beispielsweise bestehen zwei Entitäten für das Lösungselementmerkmal `NominalTorque` (Nenndrehmoment), wobei die eine ein Datenmerkmal, die andere ein Individuum darstellt. Eine semantische Verbindung zwischen den beiden besteht zwar streng genommen nicht, allerdings wird hierdurch die Anfrage nach Meta-Informationen zu dem Datenmerkmal ermöglicht. Denn bei dieser kann zwischen den beiden nicht unterschieden werden. Je nach Art der Anfrage erfolgt die Antwort auf Grundlage der modellierten Information für die jeweils „zuständige“ Entität. Letztere ergibt sich aufgrund der Position innerhalb des jeweiligen Tripels (Subjekt, Prädikat oder Objekt)[3]. Für den Nutzer entsteht genau die gewünschte Funktionalität, ohne dabei die Grenzen der Sprachebene DL zu überschreiten. Allerdings bedarf es bei der Wissensmodellierung u. U. größerer Sorgfalt.

5.2 Integration des Lösungswissens in den Entwurf

Bild 5-5 zeigt den Ablauf bei der Integration des Lösungswissen in den Entwurfsprozess in stark vereinfachter, komprimierter Form (vgl. Kapitel 3). Er beginnt in der Konzipierungsphase mit der lösungsneutralen Spezifikation von Anforderungen und Funktionen des zu entwickelnden Systems. Auf dieser Basis wird nach Lösungsmustern gesucht, welche die Funktion erfüllen und gleichzeitig den Anforderungen gerecht werden. So wird der Lösungsraum aufgespannt. Durch Kombination ergeben sich verschiedene Lösungsvarianten, deren Struktur und interne Wechselwirkungen durch die Wirkstruktur beschrieben werden. Die hierin modellierten Systemelemente stehen miteinander über Flüsse in Verbindung und nutzen vorhandene Lösungsmuster. Zum Beispiel beinhaltet die Wirkstruktur der Delta-Roboter drei Systemelemente für die drei Antriebe (vgl. Bild 3-5 auf Seite 68), die alle dasselbe Lösungsmuster `Servoantrieb` nutzen. Die Ausprägung des Musters ist in diesem speziellen Fall identisch, wird sich jedoch im Allgemeinen vielfach unterscheiden (wie z. B. bei den Antrieben des Spiralkneters). Die verschiedenen Lösungsvarianten werden anschließend mithilfe idealisierter Simulationsmodelle ausgelegt, analysiert und bewertet. Dazu werden die den verwendeten Lösungsmustern zugehörigen Lösungsmustermodelle genutzt. Die auszuarbeitende prinzipielle Lösung und die Verantwortlichkeiten werden anschließend in der

[3] Man kann hier evtl. einen Vergleich zur Substantivierung von Verben anstreben. Auch dort ergibt sich die Funktion (und damit die Groß- und Kleinschreibung) aus dem Kontext. Bsp.: „Der Antrieb *dreht* mit 800 rpm. Das *Drehen* wird gemessen in rpm.“

sog. Prinziplösung festgelegt. Im Zuge der Ausarbeitung werden dann konkrete Lösungselemente für die Bestandteile des Systems gesucht und ausgewählt. Diese beinhalten detaillierte Lösungselementmodelle des spezifischen Verhaltens, die im Anschluss dazu genutzt werden, die Funktionsfähigkeit im Detail zu überprüfen und das übrige System auszulegen.

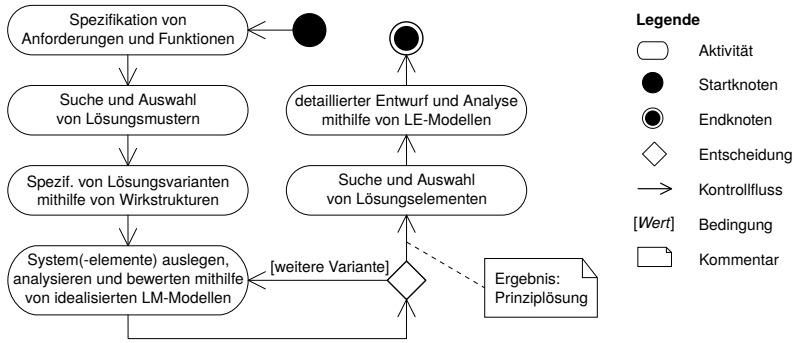


Bild 5-5: Vereinfachter Ablauf der Integration von Lösungswissen als UML-Aktivitätsdiagramm

In diesem Abschnitt soll anhand der Suche und Auswahl von Lösungselementen dargestellt werden, wie das mithilfe der erläuterten Ausdrucksmittel beschriebene, konkrete Lösungswissen in den Entwurfsprozess integriert werden kann. Die Suche und Auswahl von Lösungselementen ist ein iterativer Prozess, bei dem sukzessive ein Lösungselement nach dem anderen ausgewählt wird. Die für die Auswahl eines zu entwickelnden Systemelements zuständige Disziplin und ggf. auch eine Reihenfolge, wird in der Prinziplösung festgelegt. Trotzdem wird es Lösungselemente geben, die Einfluss auf die Ausarbeitung mehrerer unterschiedlicher Disziplinen haben und deshalb nicht losgelöst von diesen ausgewählt werden können. Beispiele hierfür sind u. a. die Antriebe der Delta-Roboter. Zumeist handelt es sich hierbei um physikalische Lösungselemente. Es sind allerdings auch Szenarien denkbar, in denen z. B. die Auswahl von Softwaretechnik-Lösungslementen (wie SPS-Funktionsbausteine) Einschränkungen an die Regelung und die Hardware nach sich zieht. Übergreifend relevanten Lösungselementen kommt daher besondere Bedeutung zu. Sie sollten möglichst vor anderen Lösungselementen und nach erfolgter disziplinübergreifender Koordination ausgewählt werden. Anschließend muss die Auswahl dokumentiert, an die betroffenen Entwickler kommuniziert und das Systemverhalten im Zuge des Analyse-Schritts überprüft werden (vgl. Kapitel 3). Andere, rein fachdisziplinspezifische Lösungselemente (z. B. Wellen, Flansche, Kabel) sind weniger kritisch und können unabhängig festgelegt werden.

Für die grundsätzliche Vorgehensweise und die semantische Suche an sich ist diese Klassifikation jedoch unerheblich.

Nachfolgend wird am Beispiel der Delta-Roboter-Antriebe gezeigt, wie mithilfe des semantisch aufbereiteten Wissens Lösungselemente gefunden und in den Systementwurf integriert werden können. Das Vorgehen steht stellvertretend für weitere, semantisch aufbereitete Lösungselemente. Zwar können die vorgestellten semantischen Technologien für den Entwickler wertvolle Hilfsmittel darstellen, es wird jedoch auch deutlich werden, dass seine Expertise für die Formulierung von Suchkriterien und die letztendliche Auswahl grundsätzlich weiterhin nötig bleibt.

5.2.1 Suchkriterien für Lösungselemente

Den Ausgangspunkt für die Suche nach Lösungswissen im Allgemeinen und Lösungselementen im Speziellen bilden möglichst präzise Suchkriterien. Neben technischen Kriterien kann es auch wirtschaftliche oder unternehmenspolitische Gründe für die Auswahl geben. Im Folgenden wird jedoch ausschließlich die Suche nach der Lösung betrachtet, die sich unter technischen Gesichtspunkten als optimal erweist. Als Ausgangspunkt für die Suche dienen die verwendeten Lösungsmuster, deren Parameter mithilfe der idealisierten Modelle (vgl. Kapitel 4) ausgelegt wurden. Letztere ermöglichen es als quantitative Suchkriterien, die Menge der Lösungselemente zu filtern, die zu einem bestimmten Lösungsmuster gehören. Darüber hinaus können auch aus den Simulationsverläufen Auswahlkriterien wie z. B. Lastprofile abgeleitet werden. Idealisierte Lösungsmustermodelle stellen somit eine sehr wichtige Informationsquelle für geeignete Such- und Auswahlkriterien dar, weil sie technische Anforderungen konkretisieren bzw. herleiten. Sie können in vielen Fällen aufwendige Messungen an Prototypen bzw. extra angefertigten Prüfstandsaufbauten ersetzen. Neben quantifizierbaren Kriterien muss jedoch auch berücksichtigt werden, ob die Kompatibilität einer Lösung zu den bereits ausgewählten Lösungselementen gegeben ist. „Weiche“ Kriterien wie beispielsweise eine gute Wartbarkeit können ebenfalls eine wichtige Rolle spielen. Im Zuge der überschlägigen Auslegung des dynamischen Systemverhaltens der Prinziplösung ergeben sich auch für die kooperierenden Delta-Roboter konkretere Anforderungen an mögliche Lösungselemente. Sie sind exemplarisch in der Tabelle 5-3 zusammengefasst, eine ausführliche Herleitung findet sich in [GTS14].

Zur Erläuterung der Ableitung von Suchkriterien und der darauf aufbauenden Suche werden die Roboter-Antriebe herausgegriffen. Das genutzte Lösungsmuster `Servoantrieb` ermöglicht Zugriff auf das in Bild 4-6 auf Seite 102 abgebildete idealisierte Dynamikmodell. Nach der Komposition des Grundsystems wird das Modell durch dieses Aktormodell ergänzt, sodass die idealisierte modellbasierte Analyse und Synthese (idealisierte Komposition) erfolgen kann. Mithilfe des Dynamikmodells des Systems wird bei der gewählten Schlagtrajektorie ein erforderliches

Tabelle 5-3: Konkretisierte Anforderungen an die Komponenten als Grundlage für die Auswahl der Lösungselemente, vgl. [GTS14, S. 190]

Lösungsmuster	Anforderung 1	Anforderung 2	Anforderung 3
Schlägerplatte	$m_{Schl} \leq 0{,}5\,\mathrm{kg}$	$D_{Schl} = 0{,}5\,\mathrm{m}$	
Ball	$m_B \approx 25\,\mathrm{g}$	$c_B \approx 10\,\mathrm{N/mm}$	$k_B \geq 0{,}6$
Kraftsensoren	$c_S \geq 10\,\mathrm{kN/mm}$	$0{,}05\,\mathrm{N} \leq F_S \leq 100\,\mathrm{N}$	
Delta-Roboter	$\Delta s_x \geq 20\,\mathrm{cm}$	$\Delta s_y \geq 20\,\mathrm{cm}$	$m_{Last} \approx 2{,}5\,\mathrm{kg}$
Roboter-Antriebe	$M_{\theta,\,max} = 1{,}3 \cdot 20\,\mathrm{N\,m}$	$\omega_{\theta,\,max} = 1{,}3 \cdot 10\,\mathrm{rad/s}$	
Neigemechanismus	$-10° \leq \varphi_{Neig} \leq 100°$	$-20° \leq \psi_{Dreh} \leq 20°$	$t_{Neig} \leq t_{Flug}$

maximales Antriebsdrehmoment $M_{\theta,\,max} = M_2 = 20\,\mathrm{N\,m}$ und eine maximale Winkelgeschwindigkeit von $n_2 = 10\,\mathrm{rad/s}$ ermittelt. Um in den Nennbereich gängiger Servoantriebe zu gelangen, wird ein Übersetzungsverhältnis des nachgeschalteten Getriebes von $\ddot{u} = 32$ angenommen. Zusätzlich wird ein Sicherheitsfaktor von 1,3 eingerechnet, damit sicher ausgeschlossen werden kann, dass der Bereich der Feldschwächung erreicht wird. Das antriebsseitige Moment ergibt sich demnach zu $M_1 = M_2/\ddot{u} = 0{,}813\,\mathrm{N\,m}$. Folglich wird der Parameter des Motornennmoments auf $M_N = 0{,}9\,\mathrm{N\,m}$ gesetzt. Der Parameter der Nenndrehzahl errechnet sich analog zu $\omega_N = 1{,}3 \cdot \ddot{u} \cdot 10\,\mathrm{rad/s} = 416\,\mathrm{rad/s} \approx 4000\,\mathrm{min}^{-1}$.

Weiterhin wird auf Basis des Nennmoments das Maximalmoment zu $M_{max} = 5\,\mathrm{N\,m}$ und die Drehmomentkonstante zu $K_M = 0{,}5\,\mathrm{N\,m/A}$ abgeschätzt. Hierdurch sind bereits alle notwendigen Parameter des Lösungsmustermodells bestimmt. Durch die im Modell hinterlegten überschlägigen Berechnungen ergeben sich die maximale Drehzahl $\omega_{max} \approx 1080\,\mathrm{rad/s}$ und ein Massenträgheitsmoment von $J \approx 0{,}268\,\mathrm{kg\,cm^2}$. Diese Parametrierung wird im Zuge der idealisierten Komposition mithilfe der Simulation überprüft. Bei entsprechender Auslegung der Regelung gelingt es mithilfe eines solchen Antriebs, den Delta-Roboter entlang der gewünschte Schlagtrajektorie zu verfahren. Die Parameter des Lösungsmustermodells können daher zur Einschränkung des Suchraums bei der semantischen Suche nach Lösungselementen dienen.

5.2.2 Semantische Suche

Mithilfe der gewonnenen Suchkriterien und der Information über das zugrundeliegende Lösungsmuster (`Servoantrieb`) kann nun die semantische Suche nach geeigneten Lösungselementen für die Roboter-Antriebe durchgeführt werden. Diese nutzt die in der Wissensbasis explizit und implizit vorliegenden Informationen. Es werden gezielte Suchanfragen gestellt, die als Ergebnis passende Lösungselemente und die Information über zugehörige Simulationsmodelle zurückliefern.

Der konzipierte, generelle Ablauf zur Suche und Auswahl von Lösungselementen ist in Bild 5-6 als Aktivitätsdiagramm dargestellt, er konkretisiert die entsprechende Aktivität des vorherigen Diagramms (siehe Bild 5-5). Als Erstes wird nach möglichen Lösungselementen für das gegebene Lösungsmuster gesucht. Die Menge dieser Elemente kann entweder nachgeschaltet oder gleichzeitig mithilfe einer entsprechenden SPARQL-Anfrage gefiltert werden. Hierzu werden die ermittelten Suchkriterien verwendet. Da in der Regel kein Lösungselement existiert, welches exakt die gewünschten Werte besitzt, wird man eine gewisse Abweichung von beispielsweise $\pm 20\,\%$ erlauben. Sofern die Suche erfolgreich ist, steht üblicherweise eine Auswahl von einigen wenigen, passenden Lösungselementen zur Verfügung, aus denen der Entwickler eines auswählen kann. Dabei stehen ihm alle semantisch modellierten Informationen zur Verfügung, indem bei Bedarf weitere Anfragen gestellt werden können. Um zu überprüfen, ob die Funktionsfähigkeit nach der Entscheidung für ein Lösungselement weiterhin gegeben ist, erfolgt die Anfrage nach einem zugehörigen detaillierten LE-Modell (vgl. Abschnitt 4.4). Dieses wird – sofern vorhanden – anstelle des Lösungsmustermodells in das vorhandene Gesamtsystemmodell geladen und mit den entsprechenden Parametern des Lösungselements parametriert. Hierdurch entsteht keinerlei Modellierungsaufwand und es kann anschließend direkt mit der modellbasierten Ausarbeitung des übrigen Systems fortgefahren werden.

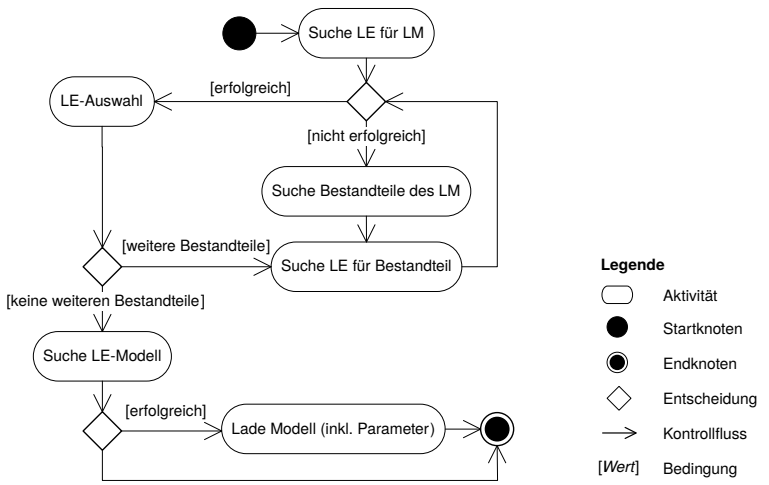


Bild 5-6: UML-Aktivitätsdiagramm für die Suche und Auswahl von Lösungselementen, vgl. [GTS14, S. 196]

Die beschriebene Suche nach Lösungselementen auf Basis eines Lösungsmusters wird jedoch nicht in jedem Fall erfolgreich sein können, weil u. U. die Suchkriterien zu eng sind oder nur wenige Lösungselemente existieren. Beispielsweise ist der Erfolg der Suche nach einem fertig konfektionierten Lösungselement `Servoantrieb` relativ unwahrscheinlich, denn die sogenannten *Kompaktantriebe* finden sich am Markt nur in sehr begrenzter Anzahl. Es ist vielmehr üblich, einen maßgeschneiderten Antrieb aus Servomotor, Servoverstärker (bzw. Frequenzumrichter) und ggf. einem Getriebe zu kombinieren. Die Suche wird folglich in der Regel nicht den gewünschten Erfolg bringen. Das Lösungsmuster `Servoantrieb` wird in diesem Fall mithilfe der modellierten `comprisesSP`-Beziehung in seine integralen Bestandteile `Servomotor` und `Servoumrichter` zerlegt und die Suche für jede dieser Komponenten solange wiederholt, bis keine Bestandteile mehr offen sind (siehe Bild 5-6).

In Quellcode 5-4 ist ein Beispiel für eine SPARQL-Suchanfrage nach Lösungselementen gezeigt. Normalerweise sollte eine solche Anfrage nicht unmittelbar in der SPARQL-Syntax formuliert werden. Damit der Entwickler keine spezifischen Kenntnis hierüber benötigt, ist sie stattdessen Ergebnis einer Konfiguration in einem Suchinterface, welches beispielsweise als Weboberfläche umgesetzt werden kann oder sich in Form einer benutzerfreundlichen grafischen Oberfläche in bekannte Entwicklungswerkzeuge integrieren lässt (vgl. Abschnitt 5.2.3). Für dieses Beispiel sind die Eingangsgrößen der Anfrage im Quellcode rot eingefärbt, die restlichen Bestandteile bleiben auch für die Suche nach anderen Lösungselementen bzw. für die Suche anhand von anderen Suchparametern bestehen. In Zeile 3 erfolgt zunächst die globale Suche nach Lösungselementen für das Lösungsmuster `Servomotor`. Mithilfe des optionalen Teils der Anfrage wird neben den URIs der Lösungselemente auch deren Label ausgegeben, falls es verfügbar ist. Um die Ergebnisse einzuschränken, wird der Modellparameter für das Nennmoment (`M_N`) verwendet. Hierfür wird im ersten Schritt mittels der Tripel in den Zeilen 7 und 8 das „Parameter-Individuum“ herausgesucht, welches zum genutzten Lösungsmustermodell gehört und darin mit `M_N` bezeichnet wird. Anschließend muss ein bzw. das Lösungselementmerkmal (`SolutionElementCharacteristic`) bestimmt werden, das Charakteristikum der gefundenen Lösungselemente ist und mit diesem Parameter korrespondiert (Zeilen 9 und 10). Dessen Wert muss sich im gegebenen Intervall von $\pm 20\,\%$ um das ermittelte Suchkriterium von $M_N = 0{,}9\,\mathrm{N\,m}$ befinden. Dies wird durch die Tripel in den Zeilen 11 und 12 sichergestellt. Selbstverständlich können durch Ergänzung entsprechender Tripel darüber hinaus gleichzeitig auch weitere Parameter zur Einschränkung verwendet werden.

Beinhaltet die Wissensbasis den in Bild 5-7 dargestellten Ausschnitt der A-Box, so liefert die Anfrage u. a. den Servomotor *AM8023E* der Firma Beckhoff zurück. Dieser wurde von Seiten des Herstellers nach eCl@ss klassifiziert und die spezi-

Quellcode 5-4: Beispiel einer SPARQL-Anfrage zur Suche von Lösungselementen vgl. [GTS14, S. 197]

```
SELECT DISTINCT *
WHERE {
 ?solutionElement se:hasSolutionPattern spa:Servomotor.
 OPTIONAL {
  ?solutionElement rdfs:label ?label
 }
 ?parameter model:isParameterOf modelLib:SPM_Servodrive1.
 ?parameter model:modelParameterAbbreviation "M_N"^^xsd:string.
 ?characteristic se:isSolutionElementCharacteristic
   ?solutionElement.
 ?characteristic entime:correspondsTo ?parameter.
 ?solutionElement ?characteristic ?value.
 FILTER (?value > 0.72 && ?value < 1.08)
}
```

fizierten Merkmale – inkl. des Nennmoments von 1,0 N m – wurden angegeben (vgl. Tabelle 5-1 auf Seite 134). Aus den beispielsweise tabellarischen Angaben wurden die entsprechenden Tripel zur Beschreibung in OWL bzw. RDF generiert. Das Ergebnis wird dem Nutzer durch das jeweilige Suchinterface dargestellt und ggf. erläutert (Erklärungskomponente). Das Diagramm in Bild 5-7 stellt die für die Beispielanfrage wichtigen Beziehungen dar, welche mithilfe der beschriebenen T-Box-Konstrukte modelliert werden.

Auf der linken Seite des Bildes sind die Lösungsmuster- und Modell-Individuen zu sehen. Den Ausgangspunkt bilden die Individuen für das Lösungsmuster `Servodrive`, für das Lösungsmustermodell `SPM_Servodrive1` und für dessen Modellparameter `M_N`. Ersteres ist mit dem Individuum `SPM_Servodrive1` verknüpft, das über `hasParameter` mit seinen Parameter-Individuen verbunden ist. Mittels der Beziehung `comprisesSP` ist modelliert, dass das Lösungsmuster `Servodrive` die Lösungsmuster `Servomotor` und `Servoamplifier` beinhaltet. Sowohl `Servomotor` als auch `Servodrive` verfügen über das Lösungsmustermerkmal `NominalTorque`, welches zum einen mit dem Modellparameter `M_N`, zum anderen mit dem Lösungselementmerkmal `NominalTorque` korrespondiert (`correspondsTo`). Auf der rechten Diagrammseite ist ein kleiner Teil der Lösungselementklassifikation für Aktoren zu sehen, in die sich die eCl@ss-Klasse `27022602` für Servomotoren integriert. Das Lösungselement `AM8023E` wird durch ein Individuum der Klasse `27022602` abgebildet. Die durch Restriktionen festgelegten Datenmerkmale der übergeordneten Klassen werden für das Individuum mit den konkreten Werten des Servomotors belegt. Einige dieser Charakteristika sind in Bild 5-7 als Attribute dargestellt. Weiterhin wird dem Individuum über die

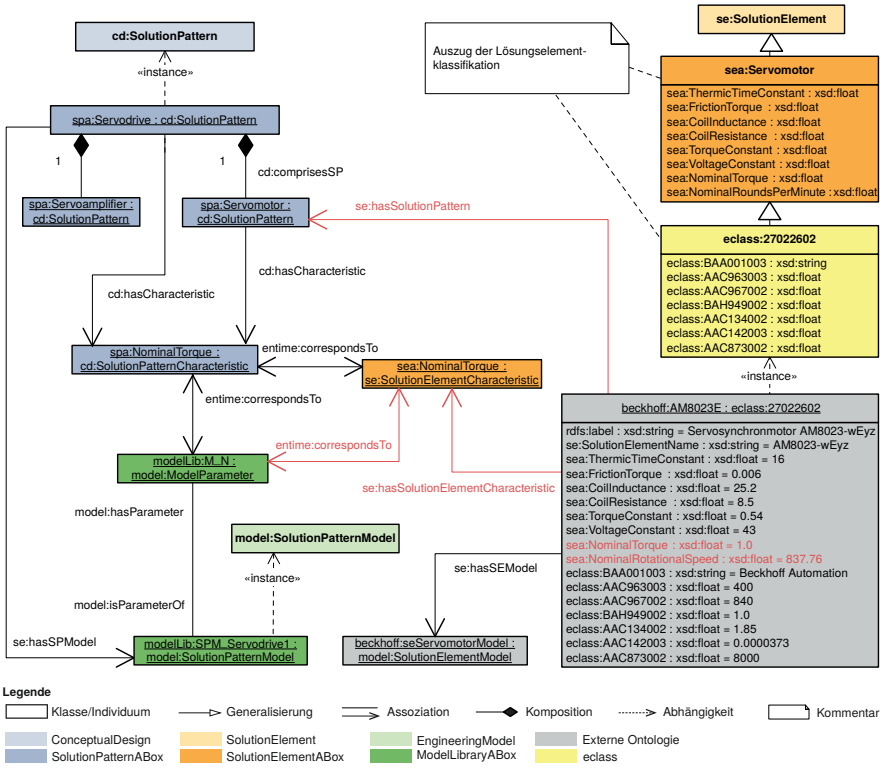


Bild 5-7: Ausschnitt relevanter Klassen und Individuen für die Suche nach einem Servomotor (Beispiele inferierter Beziehungen und Merkmale sind in rot dargestellt.) [GTS14, S. 199]

Beziehung `hasSEModel` ein entsprechendes Lösungselementmodell des Herstellers zugeordnet. Beide durch den Hersteller bereitgestellten Individuen sind in einem eigenen Namensraum (`beckhoff:`) modelliert. In dieser Ontologie werden die Lösungselement-Individuen zunächst in unternehmensspezifischen Terminologien modelliert, gleichzeitig leistet sie jedoch auch die Integration in die vorgestellten Ontologie, da die semantischen Zusammenhänge zwischen den Repräsentationen allgemeingültig abgebildet sind.

Aus den explizit modellierten Größen und Beziehungen wird durch den Reasoner weiteres, implizites Wissen inferiert. Beispiele hierfür sind in Bild 5-7 in Rot eingezeichnet. Mithilfe von Restriktionen ist beispielsweise hinterlegt, dass alle Individuen der Lösungselementklasse `Servomotor` die Verbindung `hasSolutionPattern`

zu dem Lösungsmuster `Servomotor` haben. Dies trifft folgerichtig somit auch auf den betrachteten Servomotor `AM8023E` zu. Quellcode 5-5 zeigt die Tripel dieser Restriktion. Auch die Verbindung zu dem Merkmal `NominalTorque` über das Objektmerkmal `hasSolutionElementCharacteristic` wird inferiert, denn das Merkmal `correspondsTo` ist, wie beschrieben, ein transitives Merkmal. Die Korrespondenz zwischen dem Modellparameter `M_N` und dem Lösungselementmerkmal `NominalTorque` ergibt sich folglich logisch, auch wenn sie nicht explizit angegeben ist. Denn beide korrespondieren mit demselben Lösungsmustermerkmal.

Quellcode 5-5: Beispiel für Restriktionen zur Vererbung von Verknüpfungen

```
sea:Servomotor rdf:type owl:Class;
               rdfs:subClassOf se:SolutionElement,
                     [ rdf:type owl:Restriction;
                       owl:onProperty se:hasSolutionPattern;
                       owl:hasValue spa:Servomotor
                     ] ,
                     ...
```

Nachdem aus der Menge der möglichen Lösungselemente eines ausgewählt wurde, erfolgt die Suche nach dem zugehörigen detaillierten Simulationsmodell (vgl. Abschnitt 4.4 auf Seite 119). Die entsprechende Anfrage verfolgt die mit `hasSEModel` bezeichnete Kante. Im Falle des Motors `AM8023E` werden das Lösungselementmodell `seServomotorModel` sowie die für diesen Motor gültigen Modellparameter ausgegeben. Durch die beschriebene Modellierung des Lösungswissens wird somit erreicht, dass erstens eine Verbindung zwischen Modellen, Lösungsmustern und Lösungselementen besteht, zweitens werden Modellparameter und LE-Merkmale verknüpft. Beides gelingt trotz unterschiedlicher Terminologien, Abstraktionsgrade, zugrundeliegender Denkweisen und Klassifizierungen. Hierdurch wird die Suche nach geeigneten Lösungselementen, dem zugehörigen Lösungselementmodell und dessen Parametern ermöglicht.

5.2.3 Prototypische Umsetzung und Werkzeugunterstützung

Entscheidend für die Akzeptanz und die Nutzung semantischer Technologien zur Suche von Lösungselementen ist, dass der Zugriff auf das abgelegte Wissen für den Entwickler möglichst unkompliziert ist. Detaillierte Kenntnisse über die Wissensmodellierung oder die Syntax von SPARQL-Anfragen dürfen keine Voraussetzung sein. Das Ziel muss es daher sein, die semantische Suche nahtlos in den Entwicklungsprozess, die Entwicklungsumgebung und bekannte CAE-Tools einzubinden. Auch sollten die Ergebnisse in den jeweiligen Terminologien präsentiert werden. Für die einzelnen Entwicklungsschritte des Vorgehensmodells (vgl. Kapitel 3) werden daher geeignete Anfragen vorbereitet. Weiterhin wurde eine

prototypische *Ontologie-Schnittstelle* inkl. graphischer Oberfläche für das Simulationswerkzeug Dymola entwickelt. Sie besteht aus diversen Modelica-Skripten und einem eingebetteten *C++ Plug-in*. Bild 5-8 stellt den Ablauf dar. Die Informationen über Modellparameter werden mithilfe von Modelica-Skripten extrahiert. Hierzu muss der Nutzer das zugrundeliegende Modell und das betreffende Systemelement angeben. Anschließend werden die Werte für alle durch Annotation (vgl. Quellcode 4-1 auf Seite 99) gekennzeichneten Parameter extrahiert. Hierzu wird die sog. *ModelManagement* Bibliothek verwendet [Das14].

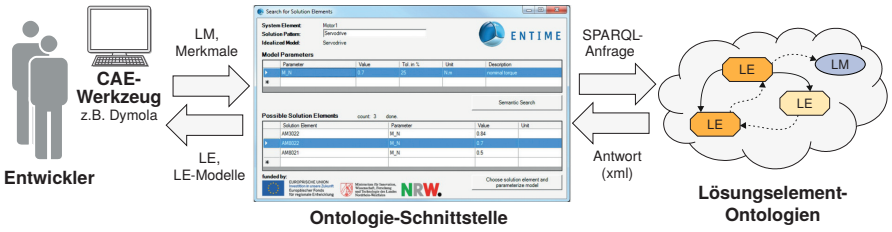


Bild 5-8: Ontologie-Schnittstelle zum Zugriff auf Lösungswissen aus CAE-Werkzeugen, vgl. [GTS14, S. 238]

Die Parameter werden an das eingebettete Plug-in übergeben, welches die Konfiguration der Anfragen auf Basis der idealisierten LM-Modelle und der Nutzereingaben für bspw. die Toleranz bzgl. der Suchkriterien übernimmt. Die erzeugte SPARQL-Anfrage wird an einen *Joseki*-Server gesendet, der über die oben beschriebene Wissensbasis und den Reasoner *Pellet* verfügt (vgl. Abschnitt 2.4.5 auf Seite 46). Die Antwort im XML-Format wird geparst und die Ergebnisse werden in der graphischen Oberfläche tabellarisch dargestellt. Der Entwicklungsingenieur kann anhand der Merkmale eines der angezeigten, passenden Lösungselemente auswählen. Für dieses kann im nächsten Schritt nach einem geeigneten, detaillierten Lösungselementmodell gesucht werden. Das Modell wird mithilfe der in der Wissensbasis gespeicherten URL aus der Datenbank heruntergeladen. Durch weitere Modelica-Skripte wird es automatisch parametriert und anstelle des idealisierten Lösungsmustermodells in das Dynamikmodell des Systems eingesetzt.

Der beschriebene Vorgang wird gemäß der zuvor definierten Reihenfolge solange schrittweise wiederholt, bis für alle Systemelemente adäquate Lösungselemente ausgewählt wurden. Auf Basis des detaillierten Dynamikmodells kann jeweils nach der Auswahl eines Lösungselements die Systemanalyse im Zuge der disziplinübergreifenden Koordination erfolgen. Der Modellierungsaufwand hierfür wird durch die Wiederverwendung der Lösungselementmodelle und durch den automatischen Austausch deutlich reduziert. Um den Aufwand des Wissensingenieurs zu reduzieren, sind verschiedene Skripte in der Programmiersprache *Python*

entstanden, die Ontologien aus tabellarischen Darstellungen der eCl@ss- bzw. Aktor-Klassifikation generieren können. Darüber hinaus besteht die Möglichkeit, XML-Daten als Grundlage zu verwenden (vgl. Abschnitt 6.3.2).

6 Semantische Aufbereitung und Nutzung von Systemwissen

In diesem Kapitel soll am Beispiel des intelligenten Kneters (vgl. Abschnitt 3.2 auf Seite 79) erläutert werden, wie Systemwissen semantisch beschrieben, zur Verfügung gestellt und im Entwicklungsprozess genutzt werden kann. Dies erfolgt in Einklang mit den in Kapitel 5 beschriebenen Ontologien. So wird einerseits erreicht, dass entsprechend verfügbares Lösungswissen integriert werden kann. Das entstehende System bzw. dessen Systemelemente können andererseits nach der Entwicklung direkt als neues semantisch aufbereitetes Lösungswissen genutzt werden. Es entsteht ein Wissensmodell des Systems, d. h. ein zentrales, semantisch definiertes Abbild der Systemzusammenhänge, welches im Laufe der Entwicklung gepflegt und als verlässliche Informationsquelle genutzt wird. Das Modell führt das Systemwissen und die Entwicklungsergebnisse der beteiligten Disziplinen zusammen, macht explizites und implizites Systemwissen verfügbar und zeigt Abhängigkeiten auf (vgl. Bild 6-1). Hierdurch wird die Kommunikation zwischen den Fachdisziplinen und ggf. auch mit Zulieferern verbessert. Das sog. Systemwissensmodell (oder semantische Systemmodell) stellt insbesondere in der disziplinübergreifenden Koordination ein wichtiges Hilfsmittel dar (vgl. Abschnitt 3.1.4 auf Seite 74) und erweitert Umfang und Ausdrucksstärke von bisherigen Systemmodellen, die z. B. in SysML modelliert werden.

Bei dem föderativen Ansatz wird gleichzeitig die Konsistenz gesichert, indem die spezifischen Modelle und Entwicklungsartefakte, welche zur Bearbeitung von Teilaspekten des Systems in spezialisierten Tools herangezogen werden, mithilfe der Informationen des Systemwissensmodells aktuell gehalten werden. Um dies zu erreichen, müssen die folgenden Aspekte im Systemwissensmodell Berücksichtigung finden. Sie orientieren sich grob an den Diagrammklassen der SysML (vgl. Abschnitt 2.3.2 auf Seite 25 bzw. [ERZ14, S. 90]).

- **Strukturinformationen:** Die Ontologien geben Auskunft über die Bestandteile, d. h. die Systemelemente des zu entwickelnden Produkts. Es können Fragen zur Verknüpfung und Modularisierung der Systemelemente beantwortet werden. Als Informationsquelle dient die Wirkstruktur (vgl. Abbildung 3-5 auf Seite 68 und Abbildung 3-12 auf Seite 82).
- **Eigenschaften:** Das Systemwissensmodell sammelt charakteristische Eigenschaften/Parameter des Systems und systemrelevante Merkmale der Systemelemente. Dies umfasst sowohl quantitative und textuell beschriebene Charakteristika als auch Verknüpfungen zu weiteren Entwicklungsartefakten.

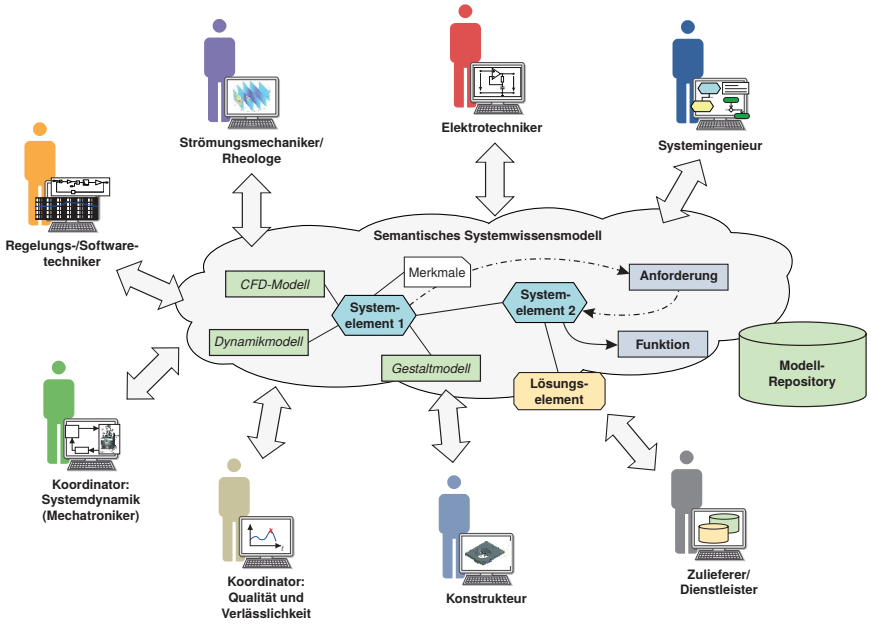


Bild 6-1: Prinzip des Systemwissensmodells

- **Modellrepräsentationen:** Es muss abgebildet werden, welche disziplinspezifischen und disziplinübergreifende Modelle existieren. Sie beschreiben u. a. die Gestalt und das (dynamische) Verhalten. Die nötigen Informationen umfassen die in Tabelle 4-1 auf Seite 98 aufgeführten Punkte. Da in den unterschiedlichen Disziplinen verschiedene Werkzeuge zum Einsatz kommen, ist darüber hinaus Wissen über diese erforderlich. Weiterhin muss die Möglichkeit bestehen, Modelle mit Systemelementen und Modellparameter mit Systemeigenschaften zu verknüpfen.
- **Ziele:** Zur Nachverfolgung und Eigenschaftsabsicherung ist es erforderlich, dass die spezifizierten Ziele, also die Systemanforderungen und Funktionen in geeigneter Form abgebildet und verknüpft werden können.

Im folgenden Abschnitt werden zunächst die Sprachkonstrukte zur Modellierung des Systemwissens eingeführt (T-Box), ehe im Abschnitt 6.2 das Systemwissensmodell des intelligenten Kneters ausschnittsweise dargelegt wird (A-Box, vgl. Abschnitt 2.4.1 auf Seite 30). Für beide werden zusätzliche Namensräume eingeführt, die in Tabelle 6-1 aufgelistet und erläutert sind. Analog zum vorherigen Kapitel sind sie in den nachfolgenden Diagrammen durch die ebenfalls dargestellte Farbe

hervorgehoben. In Abschnitt 6.3 wird anschließend der Informationsaustausch mit dem Systemwissensmodell thematisiert, der wie zuvor möglichst keine detaillierten Kenntnisse der in den Abschnitten 6.1 und 6.2 beschriebenen Ontologien und technischen Hintergründe voraussetzen soll. Durch eine entsprechende Werkzeuganbindung (z. B. an Simulationstools oder ein SysML-Modellierungswerkzeug) soll erreicht werden, dass der Aufwand für die Pflege des Systemwissensmodells durch einen Wissensingenieur möglichst gering bleibt. Das Kapitel schließt mit der Erläuterung der prototypischen Umsetzung dieser Integration ab, welche die prinzipielle Machbarkeit des dargestellten Vorgehens zeigen soll.

Tabelle 6-1: Zusätzliche Abkürzungen für Namensräume

Präfix	Erläuterung	Farbe
sysEL:	Beschreibung übergeordneter und allgemeingültiger Zusammenhänge zwischen Systemelementen (T-Box) *http://entime.uni-paderborn.de/ontology/SystemElement#*	
if:	T-Box-Konstrukte zur Modellierung von Schnittstellen *http://entime.uni-paderborn.de/ontology/Interface#*	
dac:	Wissen über die disziplinspezifischen Konzepte und Denkweisen sowie deren Zusammenhänge (T-Box) *http://entime.uni-paderborn.de/ontology/DisciplineAndConcept#*	
tool:	Beschreibung der verwendeten Werkzeuge (T-Box) *http://entime.uni-paderborn.de/ontology/EngineeringTool#*	
signal:	Ausdrucksmittel für Referenzsignale (T-Box) *http://entime.uni-paderborn.de/ontology/Signal#*	
kneader:	Systemwissensmodell des intelligenten Kneters (A-Box) *http://entime.uni-paderborn.de/Systems/IntelligentKneader#*	

6.1 Konstrukte zur Modellierung von Systemwissen

Um den dargelegten Zweck des Systemwissensmodells erfüllen zu können, bedarf es weiterer Ausdrucksmittel im Vergleich zur Repräsentation von Lösungswissen, die jedoch wiederverwendet und ergänzt werden kann. Die Bestandteile des Systems – die Systemelemente – sowie ihre Merkmale und Schnittstellen müssen beschreibbar sein. Sie hängen zusammen mit Funktionen und den Anforderungen an das System. Für die Beschreibung der verschiedenen Modelle des Systems bilden die zuvor beschriebenen Konstrukte eine gute Grundlage. Bild 6-2 zeigt einen Ausschnitt der T-Box des Systemwissensmodells, die man nach Strahringer auch als „Meta-Systemwissensmodell“ bezeichnen kann.

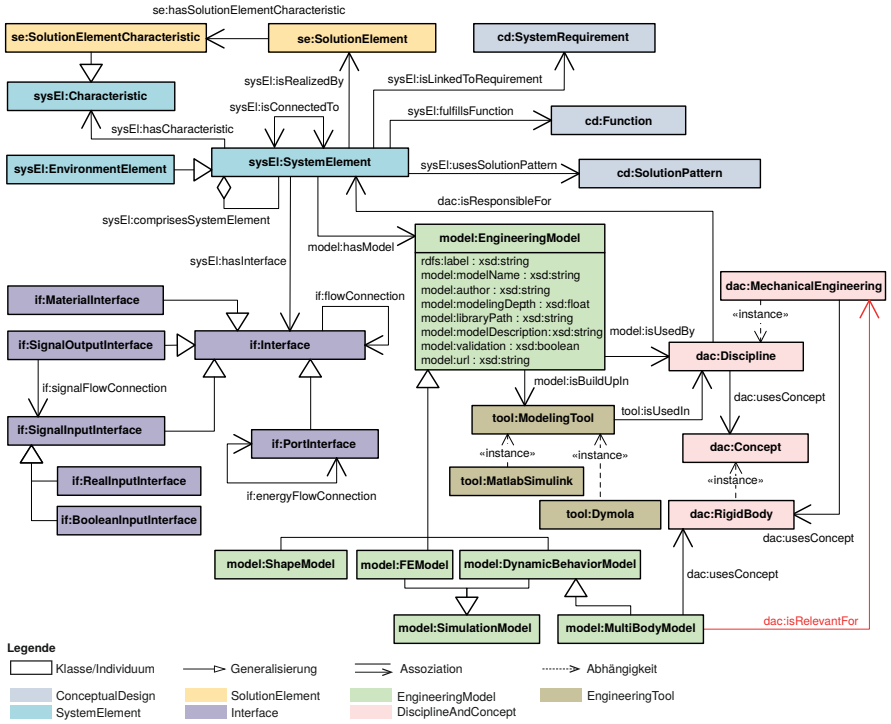


Bild 6-2: Ausschnitt der T-Box des Systemwissensmodells, dargestellt als UML-Klassendiagramm

Die System- und Umfeldelemente sowie die Flüsse zwischen ihnen bildet die Wirkstruktur ab. Sie dient als Basis, weshalb die Ontologie *SystemElement* die übergeordnete Klasse `SystemElement` einführt. Individuen dieser Klasse können miteinander in Verbindung stehen (`isConnectedTo`) und andere Systemelemente beinhalten (`comprisesSystemElement`). Mithilfe der Beziehungen `usesSolutionPattern` kann ausgedrückt werden, dass ein Systemelement ein Lösungsmuster verwendet. Ein Systemelement erfüllt eine bestimmte Funktion innerhalb des Systems. Die Modellierung dieses Zusammenhangs wird durch das Merkmal `fulfillsFunction` ermöglicht. Außerdem lässt sich durch die Verbindung `isLinkedToRequirement` eine Verknüpfung mit Systemanforderungen beschreiben. Umfeldelemente wiederum werden als spezielle Form von Systemelementen modelliert. Die Verknüpfungen zwischen den Elementen geschehen mithilfe der Schnittstellen (`Interface`). Diese können unterschieden werden in Material- und Port-Schnittstellen sowie Singaleingänge und -ausgänge. Individuen der Klasse `SignalOutputInterface`

können wie die der Klasse `SignalInputInterface` darüber hinaus nach ihrem Datentyp differenziert werden. Sie werden über das Objektmerkmal `signalFlowConnection` verbunden. Energie fließt über Port-Schnittstellen, Material über Materialschnittstellen. Während der Energiefluss immer bidirektional fließt und dementsprechend als symmetrisches Merkmal `energyFlowConnection` modelliert wird, kann das Merkmal `signalFlowConnection` nur von einem Ausgang zu einem Eingang gezogen werden. Die Rückrichtung wird durch das inverse Merkmal `inverseSignalFlowConnection` dargestellt.

Im „Meta-Systemwissensmodell“ werden darüber hinaus übergeordnete, systemunabhängige Regeln mittels SWRL abgebildet. So leitet die nachfolgende Regel die `isConnected`-Beziehung zwischen zwei Systemelementen her, sofern diese Schnittstellen besitzen, die durch die hierarchisch übergeordnete Flussbeziehung `flowConnection` verbunden sind (vgl. Quellcode 5-3 auf Seite 141).

$$\begin{aligned} &\mathit{SystemElement(?a)} \wedge \mathit{SystemElement(?b)} \wedge \mathit{hasInterface(?a,\ ?if1)} \wedge \\ &\quad \mathit{hasInterface(?b,\ ?if2)} \wedge \mathit{flowConnection(?if1,\ ?if2)} \\ &\qquad \rightarrow \mathit{isConnectedTo(?a,\ ?b)} \end{aligned}$$

Systemelemente werden durch verschiedene Modelle beschrieben. Die bereits bekannte semantische Beschreibung eines Modells kann, ähnlich wie bei Lösungselementen, über die Beziehung `hasModel` verknüpft werden. Die Klasse `EngineeringModel` wird weiter unterteilt, um verschiedene Modelle, die im Laufe der Entwicklung zu unterschiedlichen Zwecken aufgebaut werden, besser unterscheiden zu können. Weiterhin ist relevant, in welchem Werkzeug das Modell aufgebaut wurde, damit beispielsweise eine Verknüpfung der Teilmodelle zur disziplinübergreifenden Koordination erfolgen kann. Hierfür steht das Objektmerkmal `isBuildUpIn` zur Verfügung. Mithilfe der Ontologie *DisciplineAndConcept* werden die Zugehörigkeiten zu Disziplinen und die genutzten Konzepte modelliert (vgl. *domain ontologies*, Abschnitt 2.4.3 auf Seite 35). Unter anderem wird mithilfe der invers-funktionalen Beziehung `isResponsibleFor` die Zuständigkeit einer Disziplin für ein bestimmtes Systemelement ausgedrückt. Außerdem wird beispielsweise abgebildet, dass die Disziplin Maschinenbau u. a. Starrkörper (`RigidBody`) zur Beschreibung und Analyse nutzt (`usesConcept`). Dieselbe Verknüpfung besteht auch zwischen den Individuen der Klasse `MultiBodyModel`, die alle MKS-Modelle vereint. Dies wird analog zu Quellcode 5-5 auf Seite 152 durch Restriktion erreicht. Mithilfe der folgenden Regel kann auf Basis des Merkmals `usesConcept` anschließend die Relevanz einer Modellrepräsentation für eine Disziplin geschlussfolgert werden. Im beschriebenen Beispiel ergibt sich dies für alle Individuen der Klasse `MultiBodyModel` (vgl. Bild 6-2).

$$\begin{aligned} &\mathit{Discipline(?d)} \wedge \mathit{Concept(?c)} \wedge \mathit{EngineeringModel(?m)} \wedge \mathit{usesConcept(?d,\ ?c)} \\ &\quad \wedge \mathit{usesConcept(?m,\ ?c)} \rightarrow \mathit{isRelevantFor(?m,\ ?d)} \end{aligned}$$

Die Beschreibung von Systemelementcharakteristika geschieht analog zur Lösungselement-Ontologie durch Meta-Modellierung mittels Punning (vgl. Abschnitt 5.1.2 auf Seite 143). So wird einerseits die Speicherung von ausdetaillierten und erprobten Systemelementen nach der Fertigstellung des Produkts als Lösungselemente erleichtert. Andererseits vereinfacht sich auch die Integration von Lösungselementen. Sie wird durch das Merkmal `isRealizedBy` ausgedrückt. Mithilfe von Regeln kann erreicht werden, dass die Verbindung zu entsprechenden Lösungselementmerkmalen und -Modellen auch für ein Systemelement hergeleitet wird, sofern dieses durch `isRealizedBy` mit einem Lösungselement verknüpft ist. Hierzu wird eine Generalisierungsbeziehung zwischen Lösungselement- und Systemelementmerkmalen definiert. Modellparameter-Individuen können wie zuvor mittels `correspondsTo` mit den Merkmalen des zugehörigen Systemelements verknüpft werden. Aufgrund der semantischen Unabhängigkeit korrespondierender Entitäten beim Punning kann der Wert eines Lösungselementmerkmals jedoch nicht automatisch für das Systemelement inferiert werden. Um diesen zu erhalten, sind spezielle Anfragen nötig, welche die entsprechenden Tripel mittels eines `CONSTRUCT`-Teils erzeugen und explizit hinzufügen können. Dieser „Nachteil“ muss jedoch in Kauf genommen werden, da alternative Ansätze, wie zum Beispiel die Nutzung der Beziehung *owl:sameAs*[1] anstelle von `isRealizedBy` u. a. zur Folge hätten, dass zwei Systemelemente, die durch dasselbe Lösungselement realisiert werden, nicht mehr voneinander unterschieden werden können. Differierende Bezeichnungen, Zugehörigkeiten zu Funktionen/Anforderungen, Zuständigkeiten etc. könnten dann nicht mehr dargestellt werden bzw. würden zu inkonsistenten Ontologien führen.

Der sorgfältige Umgang mit Referenzsignalen zu einem System, wie Messungen oder Simulationsergebnissen, ist wichtig. Ansonsten kann den Aussagen, die anhand dieser Referenzen z. B. in der Analysephase getroffen werden, nicht vertraut werden (vgl. Abschnitt 4.2 auf Seite 94). Dies trifft insbesondere auf das Beispiel des intelligenten Teigkneters zu (vgl. Abschnitt 3.2 auf Seite 81). Auch für die Auswahl eines geeigneten Modells zur Analyse ist es entscheidend, zu wissen, ob und wie es validiert wurde. Aus diesem Grund werden Ausdrucksmittel zur Beschreibung von Metadaten zu Referenzsignalen und deren Verknüpfung bereitgestellt (vgl. Bild 6-3). Sie erlauben es, semantisch zu modellieren, dass ein Simulationsmodell mithilfe eines Referenzsignals validiert (`isValidatedBy`) bzw. ein Parameter mit seiner Hilfe identifiziert wurde (`isIdentifiedBy`). Referenzsignale können entweder Messungen (`Measurement`) oder simulierte Signale (`simulatedSignal`) sein. Sie resultieren jedoch in beiden Fällen aus genau einem Anregungssignal (`ExcitationSignal`), welches mittels des funktionalen Objektmerkmals `hasExcitation` verknüpft ist. Während sich Messungen auf System-

[1] Die *owl:sameAs*-Beziehung zwischen zwei Individuen sagt aus, dass es sich in Wahrheit um ein und dasselbe Individuum handelt.

elemente beziehen (abbildbar durch `isMeasurementOf`) und durch einen Sensor aufgezeichnet werden (abbildbar durch `isMeasuredBy`), sind simulierte Signale Resultat eines Simulationsmodells (modellierbar durch `isSimulatedSignalOf`). Alle Signale können durch Datenmerkmale wie Signalname (`signalName`), Datum (`signalDate`), Abtastfrequenz (`sampleFrequency`) oder Genauigkeit (`accuracy`) charakterisiert werden.

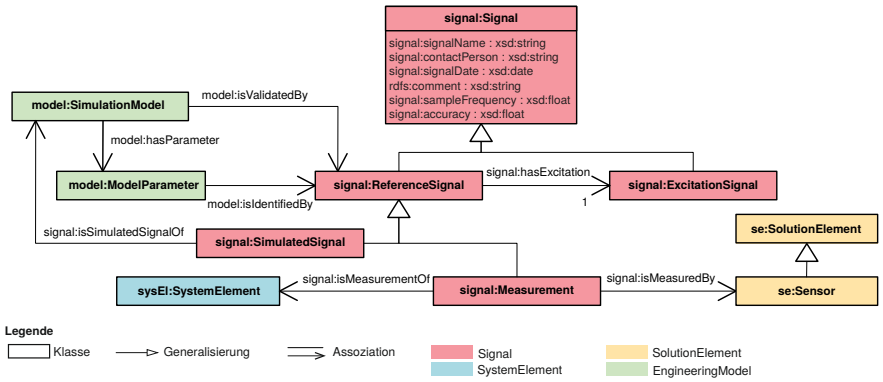


Bild 6-3: Ausdrucksmittel zur Beschreibung von Referenzsignalen

6.2 Semantisches Systemmodell

Mithilfe der dargestellten Ausdrucksmittel kann nun ein Wissensmodell für ein konkretes System aufgebaut werden. Bild 6-4 zeigt einen Ausschnitt dieser A-Box für das System des intelligenten Kneters. Die in Cyan eingefärbten Knoten stellen konkretes Wissen bzgl. des Kneters dar. Es wird im Namensraum `kneader:` modelliert. Im dargestellten Ausschnitt des Modells ist abgebildet, dass es die Systemelemente `Werkzeuglager`, `Messplatte` und `Kraftsensor2` gibt, die Bestandteil des Systemelements `Messwaage` sind (vgl. Bild 3-12 auf Seite 82)[2]. Die Menge der in der Wirkstruktur dargestellten Systemelemente wird durch die Klasse `KneterSystemElement` repräsentiert. Die hierfür nötigen Tripel sind abgekürzt in Quellcode 6-1 dargestellt (Zeilen 1–8). Darüber hinaus werden die einzelnen Systemelemente mithilfe der Zeilen 10–15 als explizit verschieden charakterisiert. In Kombination wird die *Closed World Assumption (CWA)* bzw.

[2] Der Wechsel zwischen deutschen und englischen Bezeichnungen ist in diesem Fall bewusst gewählt. Er dient einerseits dazu, die Überwindung von unterschiedlichen Terminologien zu verdeutlichen. Andererseits soll die Konsistenz zur Wirkstruktur in Bild 3-12 gewährleistet bleiben, die in deutscher Sprache modelliert wurde.

Unique Name Assumption (UNA) „erzwungen“ (vgl. Abschnitt 2.4.4 auf Seite 44). Die Menge der Systemelemente ist geschlossen und vollständig. Im Gegensatz zur Beschreibung von Lösungswissen, das prinzipiell unbegrenzt, unvollständig und erweiterbar ist, ist dieses Verhalten in diesem Fall gewünscht, denn es erzeugt restriktivere Integritätsbedingungen und ermöglicht negierte Anfragen. In diesem Punkt erinnert es dadurch an relationale Datenbanken, wenngleich natürlich alle Vorteile von Ontologien – was Inferenzmechanismen, Mehrdimensionalität etc. angeht – hierdurch nicht eingeschränkt werden.

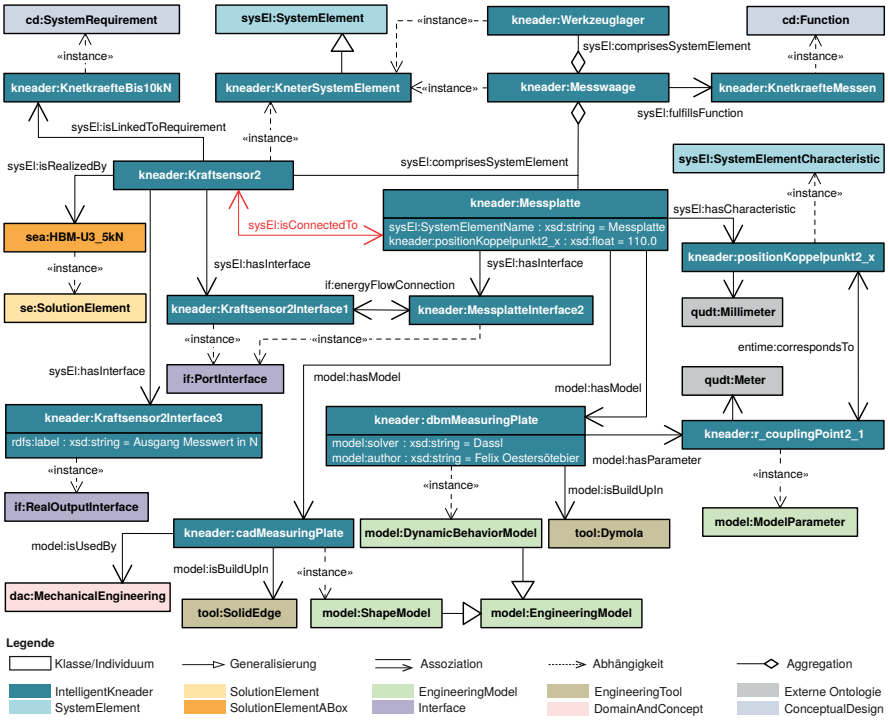


Bild 6-4: Ausschnitt der A-Box des Wissensmodells, dargestellt als UML-Klassendiagramm

Quellcode 6-1: Modellierung von explizit verschiedenen Systemelementen

```
1 kneader:KneterSystemElement rdf:type owl:Class ;
2         owl:equivalentClass [rdf:type owl:Class ;
3                              owl:oneOf (kneader:Messplatte
4                                         kneader:Messwaage
5                                         kneader:Kraftsensor1
```

```
6                                          kneader:Kraftsensor2
7                                          ... )] ;
8        rdfs:subClassOf sysEl:SystemElement .
9
10 [ rdf:type owl:AllDifferent ;
11   owl:distinctMembers ( kneader:Messplatte
12                         kneader:Messwaage
13                         kneader:Kraftsensor1
14                         kneader:Kraftsensor2
15                         ... )] .
```

In dem in Bild 6-4 dargestellten Ausschnitt ist weiterhin modelliert, dass das Systemelement `Messwaage` die Funktion „Knetkräfte messen“ erfüllt. Der `Kraftsensor2` besitzt die Port-Schnittstelle `Kraftsensor2Interface1`, die mit dem Port `MessplatteInterface2` über das Merkmal `energyFlowConnection` verknüpft ist. Aufgrund der in der T-Box hinterlegten Regel kann die Beziehung `isConnectedTo` zwischen den Systemelementen `Kraftsensor2` und `Messplatte` inferiert werden. Der Kraftsensor ist verknüpft mit der quantisierbaren Systemanforderung `KnetkraefteBis10kN`, welche die obere Grenze der zu messenden Knetkräfte spezifiziert. Unter anderem aufgrund dieser Anforderung wurde der auf Dehnungsmessstreifen basierende Sensor *U3* der Firma *HBM* als Lösungselement für die sechs Kraftsensoren ausgewählt. Dementsprechend wird das Merkmal `isRealizedBy` zwischen dem Systemelement-Individuen und dem Individuum, das dieses Lösungselement repräsentiert, gezogen. Im Beispiel befindet es sich im zentralen Namensraum der Lösungselement-A-Box. Im Sinne des Semantic Web ist es jedoch auch möglich, dass es unternehmensspezifisch und auf einem firmeneigenen Server des Anbieters semantisch modelliert ist (vgl. Abschnitt 5.2 auf Seite 144).

Darüber hinaus sind im UML-Diagramm Meta-Informationen zu zwei Modellen zu sehen. Diese sind dem Systemelement `Messplatte` zugeordnet und repräsentieren zum einen ein CAD-Gestaltmodell, zum anderen ein Dynamikmodell der Messplatte. Mithilfe des Systemelementmerkmals `positionKoppelpunkt2_x` wird die semantische Modellierung von Systemeigenschaften und deren Zusammenhang zu Modellparametern illustriert. Der Fließkommawert von 110,0 ist als Datenmerkmal des Systemelements angegeben. Er beschreibt den Abstand des Koppelpunkts vom Schwerpunkt in x-Richtung im körperfesten Koordinatensystem. Meta-Informationen des Charakteristikums, wie die Einheit `Millimeter`, sind dem durch Punning korrespondieren Individuum zugeordnet. Der konkrete Wert der Eigenschaft wird ausschließlich an dieser Stelle im Systemwissensmodell gespeichert und das dem Systemelement zugehörige Wissen dadurch zentral gesammelt. So werden mögliche Fehlerquellen, die durch zu geringe Sorgfalt bei der Modellierung mittels Punning entstehen können, minimiert. Der entsprechende Parameter des Dynamikmodells `dbmMeasuringPlate` der Messplatte ist mit dem

Systemelementmerkmal-Individuum über das Objektmerkmal `correspondsTo` verbunden, sodass hier lediglich die Referenz und nicht der konkrete Wert modelliert wird. Der Modellparameter `r_couplingPoint2_1` muss in der SI-Einheit Meter angegeben werden, was durch die Verknüpfung mit dem entsprechenden QUDT-Individuum deutlich wird. Mithilfe der QUDT-Ontologien kann der nötige Umrechnungsfaktor ermittelt werden (siehe Abschnitt 6.3.1). Neben dem Dynamikmodell, das in Dymola aufgebaut ist, besteht ein CAD-Modell im Werkzeug SolidEdge. Dieses wird von der Disziplin Maschinenbau verwendet und aufgebaut. Es besitzt einen entsprechenden Parameter für den beispielhaft betrachteten Koppelpunkt (nicht gezeigt). Durch die Modellierung der Korrespondenz bzw. durch die Referenzierung auf das zentrale Systemelementmerkmal kann sichergestellt werden, dass die verschiedenen Modellrepräsentationen der beteiligten Disziplinen konsistent bleiben. Hierzu ist es erforderlich, dass ein bidirektionaler Informationsaustausch mit dem Systemwissensmodell stattfindet.

6.3 Informationsaustausch über das Systemwissensmodell

Das semantische Systemmodell stellt einen Wissensspeicher dar, der im Laufe der Entwicklung aufgebaut, ergänzt und gepflegt wird. Dementsprechend erfolgt ein ständiger bidirektionaler Austausch zwischen den einzelnen Entwicklungswerkzeugen und dem Wissensmodell. Analog zur Suche nach Lösungselementen ist es daher erforderlich, dass sich der Informationsaustausch möglichst nahtlos in die Werkzeuge eingliedert. Bevor die prototypische Umsetzung dieser Eingliederung im Abschnitt 6.3.2 dargestellt wird, soll jedoch zunächst anhand eines Beispielszenarios die Kommunikation mit dem Systemwissensmodell mithilfe von SPARQL erläutert werden.

6.3.1 Informationsaustausch am Beispiel

Als Beispiel für den Informationsaustausch mithilfe von SPARQL soll die bereits motivierte Messung der Knetkräfte im Knetprüfstand dienen (vgl. Abschnitt 3.2 auf Seite 79). Bild 6-5 stellt die gewählte Anordnung der am Prüfstand genutzten Messwaage dar, für die vor allem die Auswahl geeigneter Kraftsensoren entscheidend ist. Die Entscheidung für einen konkreten Sensor hängt allerdings von mehreren Faktoren ab und wird von vielen beteiligten Disziplinen beeinflusst. Da es sich letztlich um die Messung dynamischer Größen handelt, fällt sie gemäß Kapitel 3 in das Aufgabengebiet des disziplinübergreifenden Koordinators der Systemdynamik.

Zuerst muss darauf geachtet werden, die Spezifikation des Systems zu erfüllen. Wie aus Bild 6-4 ersichtlich ist, können dem Systemwissensmodell Informationen

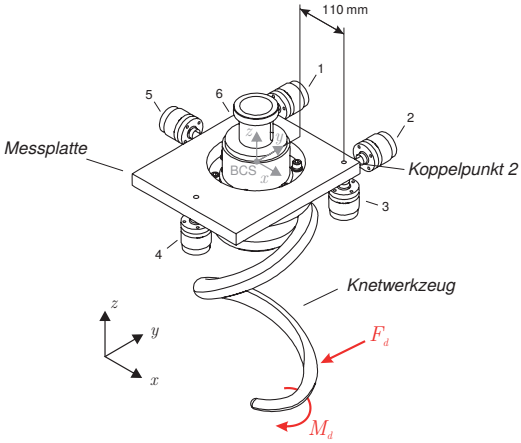


Bild 6-5: Skizze des Knetwerkzeugs inkl. der Messplatte und der Kraftsensoren 1–6[3]

über die Anforderungen und Funktionen, mit denen die Messwaage in Verbindung steht, entnommen werden. Die Anfragen hiernach sind vergleichsweise einfach und werden daher an dieser Stelle nicht gezeigt. In diesem Fall sollen die Knetkräfte in allen 6 Raumrichtungen und bis zu 10 kN gemessen werden. Dies geht aus vorherigen Belastungsanalysen sowie der in den Anforderungen spezifizierten, maximalen Mehlmenge und den Teigrezepturen hervor, welche festere Teige ausschließen. Des Weiteren ist natürlich die dynamische Beschaffenheit der Messgröße, d. h. der Knetkräfte und -momente entscheidend. Hierüber können strömungsmechanische CFD-Modelle, Partikel-Simulationen, aber auch erste Messungen an einem Prüfmuster Auskunft geben (vgl. Bild 3-13 auf Seite 84). Andererseits kann die lokal zu messende Kraft durch die Masse und Trägheit der Messplatte sowie die Lage und Anbringung der Sensoren beeinflusst werden. Diese geometrischen Größen werden durch die Konstruktionsabteilung und die dort verwendeten CAD-Gestaltmodelle geprägt. Auch die mittels FEM-Simulation abgesicherte Festigkeit hängt maßgeblich von ihnen ab. Durch den Informationsaustausch mit dem Systemwissensmodell wird die Konsistenz der erwähnten Modellrepräsentationen – aber auch der zur disziplinübergreifenden Koordination verwendeten Systemmodelle – gesichert. Für die Auswahl der Sensoren wird beispielsweise ein MKS-Modell erstellt, in das die Erkenntnisse der übrigen Untersuchungen und Simulationen in abstrahierter Art und Weise einfließen.

[3] Die Skizze wurde auf Grundlage von CAD-Daten der Firma WP Kemper GmbH erstellt.

Quellcode 6-2 zeigt beispielhaft, wie der Abstand des Koppelpunkts 2 vom körperfesten Koordinatensystem der Messplatte aus dem Systemwissensmodell abgefragt werden und damit das Dynamikmodell auf den aktuellen Stand gebracht werden kann. Dabei wird davon ausgegangen, dass das im vorherigen Abschnitt erläuterte Wissensmodell zur Verfügung steht. Ein Großteil der Anfrage bleibt auch zur Ermittlung eines beliebigen, anderen Parameters immer gleich, sodass sie leicht von einem Werkzeug generiert werden kann. Die in diesem Fall notwendigen Eingaben sind in Rot markiert. Hierbei handelt es sich um die Bezeichnungen des betreffenden Systemelements (`Messplatte`) und des jeweiligen Parameters (`r_couplingPoint2_1`). Bei der Anfrage werden zunächst das korrespondierende Systemelementmerkmal des Parameters und dessen Einheit bestimmt (Zeilen 5 und 6). Es wird überprüft, ob es sich um ein Merkmal des Systemelements `Messplatte` handelt, und der Wert ermittelt (Zeilen 7 und 8). Mithilfe dieser ersten Tripel erhält man das korrespondierende Merkmal `positionKoppelpunkt2_x` und dessen Wert von 110,0 mm (vgl. Bilder 6-4 und 6-5). Durch Zeile 9 wird nun die geforderte Einheit des Parameters ausgelesen. Dieser ist in Metern angegeben. Mithilfe der beiden nachfolgenden Zeilen wird für die Einheiten jeweils der Umrechnungsfaktor zur Basis SI-Einheit aus den QUDT-Ontologien bestimmt. Durch die `BIND`-Anweisung wird der Ausgabegröße `?result` der Wert des Parameters in Metern zugewiesen, der anhand der Umrechnungsfaktoren errechnet wurde. Zuletzt wird die Information über dessen Einheit als Abkürzung ermittelt und ausgegeben, sodass das Ergebnis der Anfrage 0,11 m lautet.

Quellcode 6-2: Beispiel einer SPARQL-Anfrage zur Ermittlung eines Modellparameters (Präfixe siehe oben)

```
1  PREFIX ...
2
3  SELECT DISTINCT ?result ?unit
4  WHERE {
5   ?char entime:correspondsTo kneader:r_couplingPoint2_1;
6         entime:hasUnit ?unit1.
7   kneader:Messplatte sysEl:hasCharacteristic ?char;
8                      ?char ?value.
9   kneader:r_couplingPoint2_1 entime:hasUnit ?unit2.
10  ?unit1 qudt:conversionMultiplier ?multiplier1.
11  ?unit2 qudt:conversionMultiplier ?multiplier2.
12  BIND(
13    (?value*?multiplier1/?multiplier2) AS ?result
14    ).
15  ?unit2 qudt:abbreviation ?unit.
16 }
```

Mithilfe der so ermittelten Werte kann die Simulation des in Bild 6-5 dargestellten Teilsystems zur Auswahl der Kraftsensoren erfolgen. Nun kann es vor-

kommen, dass sich die dynamischen Eigenschaften durch eine Veränderung des Systems verbessern lassen. Durch Variation der Abstände ergeben sich z. B. andere Hebelarme, die ggf. dazu führen, dass sich geeignetere Lösungselemente bzw. überhaupt Lösungselemente finden lassen. Eine solche Anpassung muss selbstverständlich mit der verantwortlichen Disziplin abgestimmt und anschließend an alle betreffenden Entwicklerteams kommuniziert werden. Das bedeutet, dass das Systemwissensmodell aktualisiert werden muss. Dies kann mithilfe der SPARQL-Update-Sprachbausteine erfolgen, welche die Manipulation der in einer Wissensbasis gespeicherten Tripel erlauben [GPP13]. In Quellcode 6-3 wird der beschriebene Abstand des Koppelpunkts 2 von 110 auf 120 mm gesetzt. In dem dort gezeigten `WHERE` Teil wird – analog zu vorher – das Charakteristikum des betreffenden Systemelements ermittelt, dessen Tripel verändert werden müssen. Die Einheitenumrechnung wird in umgekehrter Richtung durchgeführt. Mithilfe von `DELETE` und `INSERT` wird anschließend das alte Tripel für das Messplatten-Merkmal gelöscht und das neue eingefügt (Zeilen 3–8). Dadurch können nun auch die übrigen Modellrepräsentationen und Entwicklungsartefakte durch entsprechende Anfragen an das Systemwissensmodell aktualisiert und somit konsistent gehalten werden.

Quellcode 6-3: Beispiel einer SPARQL-Anfrage zur Aktualisierung eines Systemelementmerkmals (Präfixe siehe oben)

```
1  PREFIX ...
2
3  DELETE {
4   kneader:Messplatte ?char ?value.
5  }
6  INSERT {
7   kneader:Messplatte ?char ?result.
8  }
9  WHERE {
10  kneader:r_couplingPoint2_1 entime:hasUnit ?unit2;
11                             entime:correspondsTo ?char.
12  kneader:Messplatte sysEl:hasCharacteristic ?char;
13                     ?char ?value.
14  ?char entime:hasUnit ?unit1.
15
16  ?unit1 qudt:conversionMultiplier ?multiplier1.
17  ?unit2 qudt:conversionMultiplier ?multiplier2.
18  BIND (
19   xsd:float("0.12"^^xsd:double*?multiplier2/?multiplier1) AS
       ?result
20  ).
21 }
```

6.3.2 Prototypische Umsetzung und Werkzeugunterstützung

Als übergeordnetes Ziel wurde formuliert, dass der Entwickler keine vertieften Kenntnisse zu Ontologien oder SPARQL besitzen muss. Vielmehr soll sich die Kommunikation mit dem Systemwissensmodell möglichst nahtlos in die tägliche Arbeit eingliedern und diese vereinfachen. Zwar kann hierzu im Rahmen dieser Dissertation keine umfassende Software-Entwicklung erfolgen, dennoch soll die prinzipielle Machbarkeit beispielhaft gezeigt werden. Dazu wird ein Konzept (siehe Bild 6-6) erarbeitet und an den kritischen Stellen prototypisch umgesetzt.

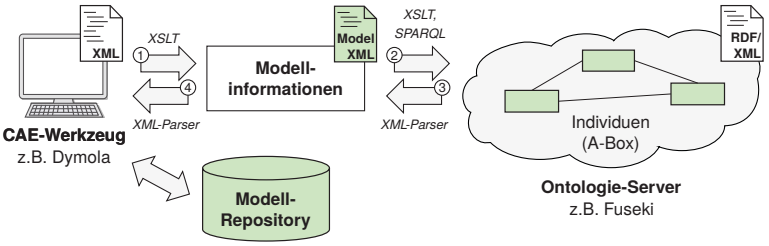


Bild 6-6: Bidirektionale Verknüpfung von Modell und Ontologie in 4 Schritten

Die Generierung bzw. Parametrierung von SPARQL-Anfragen stellt im Grundsatz keine große Herausforderung dar (vgl. auch Abschnitt 5.2.3). Einige CAE-Werkzeuge bieten bereits SQL-Schnittstellen zu Datenbanken (z. B. CATIA, Solid-Edge, MATLAB). Es soll daher an dieser Stelle um die automatische Generierung der nötigen OWL/RDF-Tripel zum Update des Wissensmodells sowie die Aktualisierung des CAE-Modells auf Basis der mittels SPARQL gewonnenen Informationen gehen. Als Beispiel dient das Werkzeug Dymola bzw. die Modellierungssprache Modelica.

Bild 6-6 deutet an, dass die Verwandtschaft von OWL und XML genutzt werden soll. Da XML eine einfache Möglichkeit bietet, Informationen rechnerunterstützt zwischen verschiedenen Quellen auszutauschen, hat sich das Format ebenfalls in vielen CAE-Werkzeugen etabliert. Für Modelica haben Pop und Fritzson mit *ModelicaXML* eine XML-Repräsentation des Codes entwickelt, die mithilfe eines Kommandozeilen-Tools[4] generiert werden kann [PF03]. Diese beinhaltet sämtliche im Modelica-Quellcode enthaltenen Informationen. Für die Übertragung in ein Systemwissensmodell bzw. ebenso für die Überführung in eine Lösungswissensbasis sind jedoch lediglich die in Tabelle 4-1 auf Seite 98 aufgeführten Informationen von Interesse. Diese werden, wie in Abschnitt 4.2 beschrieben, teilweise durch

[4] Der ModelicaXML Generator kann hier heruntergeladen werden: `https://www.ida.liu.se/~adrpo33/modelica/`

Annotationen kenntlich gemacht. Die eigentlichen Modelldateien werden im sog. Modell-Repository gespeichert.

Im ersten Schritt (siehe Bild 6-6) werden die relevanten Modellinformationen aus der ModelicaXML-Repräsentation extrahiert. Dies geschieht mithilfe einer Extensible Stylesheet Language Transformation (XSLT) [Cla99]. Dabei wird die Quelldatei sowie eine vorab definierte XSL-Datei (siehe Anhang A.3), welche die Transformationsanweisungen beinhaltet, an einen *XSL-Prozessor* gegeben[5]. Das Ergebnis ist ebenfalls eine XML-Datei, die *ModelXML* genannt wird (siehe Anhang A.2). Die hierin befindlichen Informationen sollen durch Schritt 2 in die A-Box der Ontologien überführt werden. Dazu müssen OWL-Tripel erzeugt werden, welche die in der *EngineeringModel*-Ontologie definierte Terminologie nutzen. Da auch die Ontologiesprachen auf XML beruhen, kann hierfür wiederum eine XSLT genutzt werden. Allerdings sind die nötigen Transformationen nicht trivial. Hierbei unterstützt der sog. *JXML2OWL-Mapper*[6][RRC06]. Er bietet eine graphische Oberfläche, in der die Zuordnung zwischen den XML-Tags und den T-Box-Elementen vorgenommen und anschließend die Transformationsdatei generiert wird. Die Kommunikation mit dem Ontologie-Server kann analog zu Abschnitt 5.2.3 erfolgen. Mittels SPARQL-Update können die generierten A-Box-Teile zur Wissensbasis hinzugefügt werden. Werden Informationen aus dem Systemwissensmodell abgefragt, muss die Antwort im XML-Format verarbeitet und die ModelXML-Datei aktualisiert werden (Schritt 3). Diese dient im letzten Schritt 4 als Informationsquelle für die Aktualisierung des Modells. Zu diesem Zweck ist ein generischer, objektorientierter Parser für das ModelXML-Format (siehe Anhang A.4) und ein spezifisches Modelica-Interface in Java entstanden. Ersterer stellt Funktionen zur Datenextraktion aus einer gegebenen ModelXML-Datei zur Verfügung, die durch das Modelica-Interface genutzt werden, um den Modelica-Quellcode an den entsprechenden Stellen anzupassen.

Die Schritte 1 und 4 wurden jeweils als *function* in Modelica implementiert, sodass sie über die Dymola-Benutzerschnittstelle verfügbar sind (vgl. Bild 6-6). In verschiedenen Tests konnte die Funktionsfähigkeit nachgewiesen werden. Dies gilt ebenso für die Generierung der A-Box in Schritt 3. Jedoch kann hier bisher nicht von einer reibungslosen Nutzbarkeit durch einen unerfahrenen Nutzer ausgegangen werden. Insbesondere muss die semantische Verknüpfung mit dem entsprechenden System- bzw. Lösungselement noch händisch erfolgen. Die prinzipielle Umsetzbarkeit der werkzeugunterstützten Kommunikation mit einem Ontologie-Server wurde bereits durch das in Abschnitt 5.2.3 beschriebene Plug-in gezeigt. Die Funktionsfähigkeit

[5] Bekannte XSL-Prozessoren sind z. B. Saxon (`http://saxon.sourceforge.net/`) oder Xalan (`https://xml.apache.org/xalan-j/`)

[6] Siehe auch: `http://jxml2owl.projects.semwebcentral.org`

der in diesem Kapitel dargestellten semantischen Modelle und Anfragen konnte mittels eines *Fuseki*-Servers und des Reasoners *Pellet* nachgewiesen werden.

7 Fazit und Ausblick

Abschließend soll die vorliegende Arbeit in einem Fazit zusammengefasst und hinsichtlich des identifizierten Handlungsbedarfs analysiert werden. Auf dieser Grundlage wird anschließend ein Ausblick auf mögliche, weitere Forschungsthemen gegeben.

Heutige mechatronische Systeme zeichnen sich durch ein hohes Maß an Intelligenz und Interdisziplinarität aus. Die Systeme an sich und damit auch ihre Entwicklungsprozesse sind daher von teilweise enormer Komplexität geprägt, sodass gängige Entwurfsmethodiken an Grenzen stoßen. Es bedarf daher neuer Ansätze und Methoden, welche den interdisziplinären Entwurf unterstützen, Fehleranfälligkeit reduzieren und gleichzeitig der Forderung nach kurzen Entwicklungszeiten Rechnung tragen. Stoßrichtung dieser Arbeit ist es, semantische Technologien zu nutzen, um sowohl Lösungs- als auch Systemwissen effektiv in einen konsequent modellbasierten Entwurfsprozess zu integrieren.

Nach einer Einführung in das Themengebiet (Kapitel 1) werden dazu in Kapitel 2 zunächst wichtige Grundlagen dargelegt. Es werden existierende Methoden und Vorgehensmodelle genannt und erläutert. Den anschließenden Ausführungen zu semantischen Technologien und wissensbasierten Systemen wird ein etwas größerer Bereich eingeräumt, um in diesem Themenfeld ggf. fachfremden Lesern einen Überblick über die hier wichtigen Aspekte zu geben. Mithilfe der Analyse und Bewertung verwandter Arbeiten wird der Handlungsbedarf abgeleitet. Demnach ist eine modellbasierte Entwurfsmethodik erforderlich, welche die Integration von Lösungswissen explizit vorsieht. Dieses Lösungswissen muss geeignet aufbereitet werden, um es effizient wiederverwenden zu können. Das gilt insbesondere für die Wiederverwendung von Dynamikmodellen. Des Weiteren Bedarf es der Unterstützung des Informationsaustausch zwischen den beteiligten Disziplinen während des Entwurfsprozesses. In Kapitel 3 wird das mitentwickelte Vorgehensmodell für einen solchen Entwurfsprozess anhand zweier Anwendungsbeispiele vorgestellt (vgl. [Kru, Loc]). Dies basiert auf dem V-Modell der VDI-Richtlinie 2206, kombiniert dessen Stärken jedoch mit denen der mechatronischen Komposition sowie der Spezifikationstechnik CONSENS für die disziplinübergreifende Beschreibung der Prinziplösung. Es beinhaltet insbesondere Prozessschritte zur Integration von Lösungswissen sowie die Phase der sog. disziplinübergreifende Koordination, welche den Informationsaustausch und die Konsistenz zwischen den parallelen Entwicklungen in den beteiligten Disziplinen sicherstellen soll. Gerade die Dynamikanforderungen an technische Systeme sind vielfach kritisch und erfordern eine umfangreiche modellbasierte Synthese und Analyse. Aufwand und Know-How, die in die Erstellung eines geeigneten Dynamikmodells fließen, sind mitunter immens,

weshalb ihre zielgerichtete Wiederverwendung als Lösungswissen sinnvoll ist. Die Frage nach der geeigneten Modellierung und Aufbereitung von wiederverwendbaren Dynamikmodellen beantwortet Kapitel 4 anhand verschiedener Beispiele. Durch die maschinenlesbare Annotation von Modellinformationen sowie die Zuordnung zu anderem Lösungswissen – wie Lösungsmuster oder Lösungselemente – wird die Suche und Auswahl des geeigneten Modells aus einem Repository erleichtert. Die bereitzustellenden Informationen gliedern sich in quantifizierbare und informelle Angaben zu Parametern und zum Modell im Allgemeinen. Speziell werden die Kriterien Modellierungstiefe und Modellkomplexität quantifiziert, damit eine maschinell unterstützte Auswahl entsprechend der sich erhöhenden Detaillierung im Laufe des Entwicklungsprozesses erfolgen kann.

Mithilfe der modellbasierten Untersuchungen wird eine Konkretisierung der Komponentenanforderungen erreicht und damit der Ausgangspunkt für die Suche nach den am besten geeigneten Lösungselementen geschaffen. Diese Suche erfordert weiterhin die Überwindung von unterschiedlichen, herstellerspezifischen Klassifikationen, Terminologien, Abstraktionsgraden und zugrundeliegenden Denkweisen, die nicht ohne Weiteres vergleichbar sind. Zur Überwindung dieser Differenzen werden semantische Technologien genutzt. Dabei wird davon ausgegangen, dass die Zulieferer konkretes Wissen über angebotene Lösungselemente entsprechend zur Verfügung stellen, um sich hierdurch neue Vertriebskanäle zu erschließen und auf ihr Angebot aufmerksam zu machen. Kapitel 5 zeigt, wie Lösungswissen bzw. Lösungselemente semantisch aufbereitet und in den Entwurfsprozess integriert werden können. Mithilfe der Ontologien wird eine Verbindung zwischen Modellen, Lösungsmustern und Lösungselementen sowie deren Parametern bzw. Merkmalen hergestellt. Zuletzt wird gezeigt, wie semantische Technologien genutzt werden können, um Systemwissen während der Entwicklung verfügbar zu machen (Kapitel 6). Das sog. Systemwissensmodell dient als zentrales, semantisch definiertes Abbild der Systemzusammenhänge, welches im Laufe der Entwicklung gepflegt und als verlässliche Informationsquelle genutzt wird. Dadurch kann es entscheidend zur Konsistenz, Kommunikation, Koordination sowie zum Komplexitätsmanagement beitragen. Die semantische Modellierung erfolgt im Einklang mit den Ontologien zur Abbildung von Lösungswissen, sodass das Produkt nach erfolgreich abgeschlossener Entwicklung als neues Lösungselement für andere Entwicklungen zur Verfügung gestellt werden kann. Die Modellierung und Nutzung der Ontologien wird in beiden Fällen anhand von Beispielen erläutert.

Die vorliegende Arbeit liefert einen wesentlichen Beitrag, um dem identifizierten Handlungsbedarf zu begegnen. Der vorgestellte, sequentielle Entwurfsprozess stellt für die Entwickler eine wichtige Orientierung beim konsequent modellbasierten Entwurf dar, wenngleich Iterationen unvermeidbar und z. T. auch gewünscht sind. Durch die Wiederverwendung von Lösungswissen und die Nutzung von System-

wissen können Fehler vermieden, Komplexität und Aufwand reduziert werden. Dies bedingt jedoch eine geeignete Aufbereitung von Lösungswissen (wie Lösungselemente, Dynamikmodelle), die mithilfe der präsentierten Spezifikationen nun möglich ist. Die erarbeiteten semantischen Modelle – insbesondere die jeweiligen T-Box-Anteile – sind Voraussetzung und entscheidender Wegbereiter, um Lösungs- und Systemwissen über Grenzen hinweg verfügbar zu machen. Die Machbarkeit der Konzepte wird jeweils durch eine prototypische Werkzeugunterstützung zur Integration der Lösungselementsuche sowie des föderativen Informationsaustauschs nachgewiesen. Die erarbeiteten Konstrukte zur Modellierung von System- und Lösungswissen sollten allerdings an zusätzlichen Beispielen getestet und evaluiert werden. Dazu ist es erforderlich, die bestehenden Ansätze zur Umsetzung und Werkzeugunterstützung hinsichtlich Benutzerfreundlichkeit, Leistungsfähigkeit und Robustheit zu verbessern und auszuarbeiten, sodass ein vollwertiges wissensbasiertes System entsteht (vgl. Bild 2-11 auf Seite 31). Zudem wurde der Aspekt des „Crawlers“ (vgl. Abschnitt 2.4.2 auf Seite 32) bisher nicht detailliert betrachtet. Weiterhin können die Ontologien mithilfe der SPARQL Inferencing Notation (SPIN) direkt vorgefertigte Anfragen und Regeln zur Konstruktion von Tripeln bereitstellen [KHI11], die jedoch die Entscheidbarkeit der Ontologien nicht gefährden dürfen. Um den gewünschten Nutzen der semantischen Technologien zur Wiederverwendung von Lösungswissen flächendeckend zu erreichen, ist es erforderlich, dass die A-Boxen der Wissensbasen entsprechend befüllt werden. Das gilt sowohl für Lösungsmuster und Lösungselemente als auch für Dynamikmodelle. Hier müssen Hilfsmittel und Anreize geschaffen werden. Bzgl. des Systemwissensmodells besteht die Möglichkeit, weitere Informationen zu integrieren und zu verknüpfen. Potential besteht u. a. in der Nutzung zum Versions-, Varianten- oder Produktlebenszyklusmanagement.

Im Kontext dieser Arbeit lässt sich darüber hinaus weitergehendes Forschungspotential identifizieren. Bild 7-1 zeigt das Konzept eines *Multifunktionalen Modell-Clients*, der eine ganzheitliche Unterstützung bei Aufbau und Pflege von Dynamikmodellen bieten soll (vgl. [AOT15]). Insbesondere konfiguriert, integriert und erstellt er werkzeugunabhängig Dynamikmodelle, z. B. zum Zweck der disziplinübergreifenden Koordination. Durch die Anbindung an Lösungs- und Systemwissen (z. B mithilfe der Konzepte aus den Abschnitten 5.2.3 und 6.3.2) und die Nutzung von sog. *Feature Models* [Bat05] zur Ermittlung von möglichen Modellkombinationen ist er in der Lage, folgende Funktionen zu erfüllen:

1) Konfiguration eines Dynamikmodells des Systems auf Basis der Nutzereingaben für Modellierungstiefe etc.,

2) Sicherung der Konsistenz durch Informationsaustausch mit dem Systemwissensmodell,

3) Zugriff auf wiederverwendbares Lösungswissen und Modellbibliotheken bzw. Repositories.

Das Systemwissensmodell kann weiterhin genutzt werden, um in der Informationsverarbeitung des Systems Wissen über sich selbst zu speichern. Im Kontext von *CPS* und *Industrie 4.0* scheint dies ein Ansatz zu sein, um Daten, Informationen und Wissen selbst dann über Anlagen- und Unternehmensgrenzen hinweg zu kommunizieren (vgl. [Jum15]), wenn diese Kommunikation zum Entwurfszeitpunkt nicht vorhersehbar war.

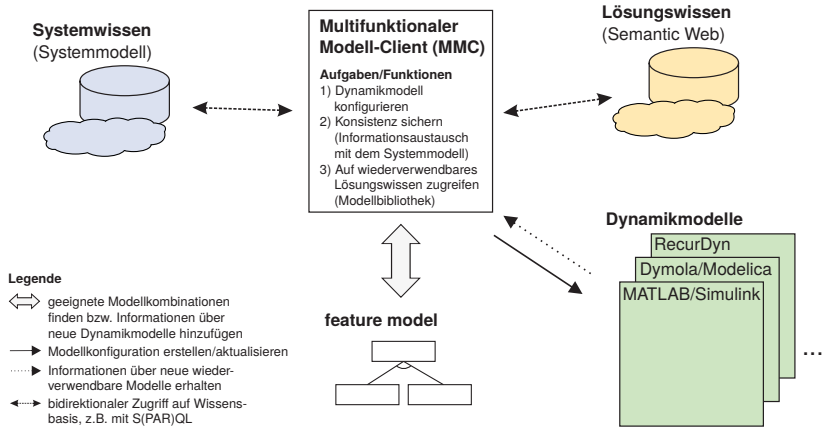


Bild 7-1: Multifunktionaler Modell-Client, vgl. [AOT15]

8 Literaturverzeichnis

[Abb12] Abburu, S.: A Survey on Ontology Reasoners and Comparison. In: *International Journal of Computer Applications* 57 (2012), November, Nr. 17

[AFT+10] Alvarez Cabrera, A.; Foeken, M.; Tekin, O.; Woestenenk, K.; Erden, M.; Schutter, B. D.; Tooren, M.; Babuska, R.; van, F. H.; Tomiyama, T.: Towards automation of control software: A review of challenges in mechatronic design. In: *Mechatronics* 20 (2010), Nr. 8, 876–886. http://doc.utwente.nl/80227/

[AIS+77] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahlking, I.; Angel, S.: *Pattern Language – Towns, Buildings, Construction.* Oxford University Press, 1st Edition, 1977

[AIS+95] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahlking, I.; Angel, S.; Czech, H. (Hrsg.): *Eine Muster-Sprache: Städte, Gebäude, Konstruktion.* Wien: Löcker Verlag, 1995

[Alt12] Alt, O.: *Modellbasierte Systementwicklung mit SysML.* Carl Hanser Verlag GmbH & Company KG, 2012

[AML14] AutomationML consortium: *AutomationML – The Glue for Seamless Automation Engineering: Whitepaper AutomationML Part 1 - Architecture and general requirements.* Version: Oktober 2014. https://www.automationml.org/o.red/uploads/dateien/1417686950-AutomationML%20Whitepaper%20Part%201%20-%20AutomationML%20Architecture%20v2_Oct2014.pdf

[Ana15] Anacker, H.: *Instrumentarium für einen lösungsmusterbasierten Entwurf fortgeschrittener mechatronischer Systeme*, Universität Paderborn, Dissertation, 2015

[AOT15] Abrishamchian, F.; Oestersötebier, F.; Trächtler, A.: Feature Model Approach For Managing Variability of Dynamic Behavior Models in Mechatronic Systems. In: *Proceedings of the ASME 2015 International Mechanical Engineering Congress & Exposition.* Houston, Texas: ASME, November 2015

[Bai08] Baier, E.: Semantische Technologien in Wissensmanagementlösungen - Einsatzpotenziale für den Mittelstand. In: *FAZIT Schriftenreihe* (2008), Nr. 13

[Bat05] Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: *Proceedings of the 9th International Conference on Software Product Lines*, Springer-Verlag, 2005, S. 7–20

[BB11] Beckett, D.; Berners-Lee, T.: *Turtle – Terse RDF Triple Language*. `http://www.w3.org/TeamSubmission/turtle/`. Version: 2011. – W3C Team Submission

[BBT+98] Breunese, A. P. J.; Broenink, J. F.; Top, J. L.; Akkermans, J. M.: Libraries of reusable models: theory and application. In: *Simulation* 71 (1998), Nr. 1, S. 7–22

[BDL+09] Binkley, D.; Davis, M.; Lawrie, D.; Morrell, C.: To CamelCase or Under_score. In: *Proceedings of 17th IEEE International Conference on Program Comprehension (ICPC)*. Vancouver, Kanada: IEEE, 2009, S. 158–167

[Bec04] Beckett, D.: *RDF/XML Syntax Specification*. W3C. `http://www.w3.org/TR/REC-rdf-syntax`. Version: 2004

[Bee94] Beerens, C.: *Zur Modellierung nichtlinearer Därnpfungs?phänomene in der Strukturmechanik*, Ruhr-Universität Bochum, Dissertation, 1994

[BF99] Berners-Lee, T.; Fischetti, M.: *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper, San Francisco, 1999

[BFB14] Barbieri, G.; Fantuzzi, C.; Borsari, R.: A model-based design methodology for the development of mechatronic systems. In: *Mechatronics* 24 (2014), Nr. 7, S. 833–843

[BG04] Brickley, D.; Guha, R. V.: *Resource Description Framework (RDF) Schema Specification*. W3C. `http://www.w3.org/TR/rdf-schema`. Version: Februar 2004

[BG14] Brickley, D.; Guha, R.: *RDF Schema 1.1, W3C Recommendation*. W3C. `http://www.w3.org/TR/rdf-schema/`. Version: Februar 2014

[BGK+12] Bauer, F.; Gausemeier, J.; Köchling, D.; Oestersötebier, F.: Simulative Absicherung mechatronischer Systeme in der frühen Phase der Produktentstehung. In: *Tag des Systems Engineering*, 2012, S. 197–206

[BGK+13] Bauer, F.; Gausemeier, J.; Köchling, D.; Oestersötebier, F.: Approach for an Early Validation of Mechatronic Systems using Idealized Simulation Models within the Conceptual Design. In: *Smart Product Engineering - Proceedings of the 23rd CIRP Design Conference*, Springer Berlin/Heidelberg, März 2013, S. 273–282

[BHL01] Berners-Lee, T.; Hendler, J.; Lassila, O.: The Semantic Web. In: *Scientific American* 284 (2001), Mai, Nr. 5, 34–43. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

[BHL+09] Bray, T.; Hollander, D.; Layman, A.; Tobin, R.; Thompson, H. S.: *Namespaces in XML 1.0 (Third Edition)*. W3C Recommendation. http://www.w3.org/TR/REC-xml-names/. Version: Dezember 2009

[Bid04] Biddulph, M.: Crawling the semantic Web. In: *XML Europe Conference*. Amsterdam, Niederlande: IDEAlliance, April 2004

[Bir06] Birkenbihl, K.: Standards für das Semantic Web. In: Pellegrini, T. (Hrsg.); Blumauer, A. (Hrsg.): *Semantic Web - Wege zur vernetzten Wissensgesellschaft*. Springer-Verlag Berlin Heidelberg, 2006, S. 73–88

[BJF08] Bi, L.; Jiao, Z.; Fan, S.: Ontology-based information integration framework for mechatronics system multi-disciplinary design. In: *6th IEEE International Conference on Industrial Informatics, INDIN 2008* IEEE, 2008, S. 831–836

[BKI08] Beierle, C.; Kern-Isberner, G.: *Methoden wissensbasierter Systeme*. 4. Vieweg und Teubner, GWV Fachverlage GmbH, 2008

[Blo72] Bloksma, A. H.: Rheology of Wheat Flour Doughs. In: *Journal of Texture Studies* 3 (1972), S. 3–17

[Blu13] Blume, T.: *Online-Wörterbuch Philosophie: Das Philosophielexikon im Internet*. Online-Quelle. http://www.philosophie-woerterbuch.de/online-woerterbuch/?tx_gbwbphilosophie_main[entry]=283&tx_gbwbphilosophie_main[action]=show&tx_gbwbphilosophie_main[controller]=Lexicon&cHash=d18ad3c6553112b231e482016c8f1d9b. Version: November 2013

[BOS00] Brandstetter, S.; Oberschelp, O.; Seuss, J.: Modellbildung. In: Gausemeier, J. (Hrsg.); Lückel, J. (Hrsg.): *Entwicklungsumgebungen Mechatronik: Methoden und Werkzeuge zur Entwicklung mechatronischer Systeme*. HNI Verlagsschriftenreihe Band 80, 2000, S. 119–154

[BP06] Blumauer, A.; Pellegrini, T.: Semantic Web und semantische Technologien: Zentrale Begriffe und Unterscheidungen. In: Pellegrini, T. (Hrsg.); Blumauer, A. (Hrsg.): *Semantic Web - Wege zur*

vernetzten Wissensgesellschaft. Springer-Verlag Berlin Heidelberg, 2006, S. 9–26

[BPS+08] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; Yergeau, F.: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126/. Version: 2008

[BR00] Beek, D. A.; Rooda, J. E.: Languages and applications in hybrid modelling and simulation: Positioning of Chi. In: *Control Engineering Practice 8*, 2000, S. 81–91

[BT96] Brooks, R. J.; Tobias, A. M.: Choosing the Best Model: Level of Detail, Complexity and Model Performance. In: *Mathematical and Computer Modeling* 24 (1996), Nr. 4, S. 1–14

[Buc14] Buckingham, E.: On physically similar systems; illustrations of the use of dimensional equations. In: *Physical Review 4* (1914)

[CBP+09] Chen, K.; Bankston, J.; Panchal, J. H.; Schaefer, D.: A Framework for Integrated Design of Mechatronic Systems. In: Wang, L. (Hrsg.); Nee, A. Y. (Hrsg.): *Collaborative Design and Planning for Digital Manufacturing*. Springer London, 2009, S. 37–70

[CCC11] CAR 2 CAR Communication Consortium Confidential (Hrsg.): *Memorandum of Understanding for OEMs within the CAR 2 CAR Communication Consortium on Deployment Strategy for cooperative ITS in Europe*. Juni 2011. – https://www.car-2-car.org

[CDD+04] Carroll, J. J.; Dickinson, I.; Dollin, C.; Reynolds, D.; Seaborne, A.; Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* ACM, 2004, S. 74–83

[CF91] Clark, K. B.; Fujimoto, T.: *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry*. Harvard Business School Press, 1991

[Cla91] Clavel, R.: *Conception d'un robot parallèle rapide à 4 degrés de liberté*, École polytechnique fédérale de Lausanne EPFL, Dissertation, 1991

[Cla99] Clark, J.: *XSL Transformations (XSLT) Version 1.0, W3C Recommendation*. http://www.w3.org/TR/1999/REC-xslt-19991116. Version: November 1999

[COV86] Cranach, M. von; Ochsenbein, G.; Valach, L.: The group as a self-active system: Outline of a theory of group action. In: *European Journal of Social Psychology* 16 (1986), Nr. 3, S. 193–229

[Czi06] Czichos, H.: *Mechatronik: Grundlagen und Anwendungen technischer Systeme*. Vieweg, 2006 (Studium Technik)

[Das14] Dassault Systèmes AB: *Dymola – Dynamic Modeling Laboratory User Manual Volume 2*. 16. Lund, Schweden, 2014

[DB15] Dhingra, V.; Bhatia, K. K.: SemCrawl: Framework for Crawling Ontology Annotated Web Documents for Intelligent Information Retrieval. In: Buyya, R. (Hrsg.); Thampi, S. M. (Hrsg.): *Intelligent Distributed Computing* Bd. 321. Springer International Publishing, 2015, S. 213–223

[DBH+12] Dziwok, S.; Bröker, K.; Heinzemann, C.; Tichy, M.: A Catalog of Real-Time Coordination Patterns for Advanced Mechatronic Systems / Fachgruppe Softwaretechnik, Heinz Nixdorf Institut, Universität Paderborn. 2012 (tr-ri-12-319). – Forschungsbericht. – `http://go.upb.de/1z37gvrc70kgo`

[DCT+11] Dentler, K.; Cornet, R.; Teije, A. ten; Keizer, N. de: Comparison of Reasoners for large Ontologies in the OWL 2 EL Profile. In: *Semantic Web* 1 (2011), S. 1–5

[Den12] Dengel, A. (Hrsg.): *Semantische Technologien*. Spektrum Akademischer Verlag GmbH, 2012

[DIN02] Norm DIN EN ISO 10303 2002. *Produktdatendarstellung und -austausch*

[DIN12] Norm DIN 4000-1 2012. *Sachmerkmal-Listen - Teil 1: Begriffe und Grundsätze*

[DJS+13] Dziwok, S.; Just, V.; Schierbaum, T.; Schäfer, W.; Trächtler, A.; Gausemeier, J.: Integrierter Regelungs- und Softwareentwurf für komplexe mechatronische Systeme. In: Gausemeier, J. (Hrsg.); Dumitrescu, R. (Hrsg.); Rammig, F.-J. (Hrsg.); Schäfer, W. (Hrsg.); Trächtler, A. (Hrsg.); Heinz Nixdorf Institut (Veranst.): *9. Paderborner Workshop Entwurf mechatronischer Systeme* Bd. 310 Heinz Nixdorf Institut, HNI Verlagsschriftenreihe, Paderborn, April 2013, S. 375–394

[Dör89] Dörner, D.: *Die Logik des Misslingens - Strategisches Denken in komplexen Situationen*. Rowohlt Verlag, 1989

[Dum10] Dumitrescu, R.: *Entwicklungssystematik zur Integration kognitiver Funktionen in fortgeschrittene mechatronische Systeme*, Universität Paderborn, Dissertation, 2010

[ECL09] eCl@ss e.V.: *eCl@ss – International standard for classifikation and description of products and services*. 2009

[EM13] Ehrlenspiel, K.; Meerkamm, H.: *Integrierte Produktentwicklung: Denkabläufe, Methodeneinsatz, Zusammenarbeit*. 5. Carl Hanser Verlag GmbH & Company KG, 2013

[ERZ14] Eigner, M. (Hrsg.); Roubanov, D. (Hrsg.); Zafirov, R. (Hrsg.): *Modellbasierte Virtuelle Produktentwicklung*. Berlin Heidelberg: Springer Vieweg, 2014

[ETI11] ETIM Deutschland e. V.: *ETIM 5.0 – Das Klassifizierungsmodell der Elektrobranche*. 2011

[Fen04] Fensel, D.: *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Second Edition. Berlin, Heidelberg: Springer-Verlag, 2004

[Föl13] Föllinger, O.; Konigorski, U. (Hrsg.); Lohmann, B. (Hrsg.); Roppenecker, G. (Hrsg.); Trächtler, A. (Hrsg.): *Regelungstechnik*. 11. VDE Verlag Berlin, 2013

[FMI14] Modelica Association: *Functional Mock-up Interface for Model Exchange and Co-Simulation*. `https://www.fmi-standard.org/`. Version: 2014

[FMS08] Friedenthal, S.; Moore, A.; Steiner, R.: *OMG Systems Modeling Language Tutorial*. `http://www.omgsysml.org/`. Version: 2008

[FMS12] Friedenthal, S.; Moore, A.; Steiner, R.: *A Practical Guide to SysML – The Systems Modeling Language*. Second Edition. Waltham, MA, USA: Morgan Kaufmann Publishers, 2012

[FPC+04] Finin, T.; Peng, Y.; Cost, R.; Sachs, J.; Joshi, A.; Reddivari, P.; Pan, R.; Doshi, V.; Ding, L.: Swoogle: A Search and Metadata Engine for the Semantic Web. In: *Proceedings of the 13th ACM International Conference on Information and Knowledge Management* ACM, ACM Press, 2004, S. 652–659

[Fri04] Fritzson, P.: *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. IEEE Press; Wiley-Interscience, 2004

[FT09] Foeken, M.; Tooren, M. van: Object-Oriented Simulation Model Generation in an Automated Control Software Development Framework. In: *Proceedings of the 19th CIRP Design Conference – Competitive Design* Cranfield University Press, 2009, S. 443–450

[FV10] Foeken, M.; Voskuijl, M.: Knowledge-based simulation model generation for control law design applied to a quadrotor UAV. In: *Mathematical and Computer Modelling of Dynamical Systems: Methods, Tools and Applications in Engineering and Related Sciences* 16 (2010), Nr. 3, S. 241–256

[Gaa10] Gaag, A.: *Entwicklung einer Ontologie zur funktionsorientierten Lösungssuche in der Produktentwicklung*, Technische Universität München, Dissertation, 2010

[GDS+13] Gausemeier, J.; Dumitrescu, R.; Steffen, D.; Czaja, A. M.; Tschirner, C.; Wiederkehr, O.: Systems Engineering in der industriellen Praxis. In: *Tag des Systems Engineering 2013 - Der Weg zu technischen Systemen von morgen.* Stuttgart: Gesellschaft für Systems Engineering e.V., 7.–9. November 2013, S. 113–122

[GFD+08] Gausemeier, J.; Frank, U.; Donoth, J.; Kahl, S.: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme des Maschinenbaus. In: *Konstruktion* (2008)

[GHJ+94] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design patterns: elements of reusable object-oriented software.* Pearson Education, 1994

[GHM+14] Glimm, B.; Horrocks, I.; Motik, B.; Stoilos, G.; Wang, Z.: HermiT: An OWL 2 Reasoner. In: *Journal of Automated Reasoning* (2014), S. 1–25

[GKP+10] Gausemeier, J.; Kaiser, L.; Pook, S.; Nyssen, A.; Terfloth, A.: Rechnerunterstützte Modellierung der Prinziplösung mechatronischer Systeme. In: *7. Paderborner Workshop Entwurf mechatronischer Systeme* Bd. 272, HNI Verlagsschriftenreihe, Paderborn, 2010, S. 53–66

[GMF+03] Gennari, J. H.; Musen, M. A.; Fergerson, R. W.; Grosso, W. E.; Crubézy, M.; Eriksson, H.; Noy, N. F.; Tu, S. W.: The evolution of Protégé: an environment for knowledge-based systems development. In: *International Journal of Human-Computer Studies* (2003), Nr. 58

[GPL+08] Gail, J.; Pöppel-Decker, M.; Lorig, M.; Eggers, A.; Lerner, M.; Ellmers, U.: *Berichte der Bundesanstalt für Straßenwesen – Fahrzeugtechnik.* Bd. F 70: *Einfluss verbesserter Fahrzeugsicherheit bei Pkw auf die Entwicklung von Landstraßenunfällen.* Bundesanstalt für Straßenwesen, 2008

[GPP13] Gearon, P.; Passant, A.; Polleres, A.: *SPARQL 1.1 Update, W3C Recommendation.* `http://www.w3.org/TR/2013/REC-sparql11-update-20130321/`. Version: März 2013

[Gra09] Graves, H.: Integrating SysML and OWL. In: Hoekstra, R. (Hrsg.); Patel-Schneider, P. F. (Hrsg.): *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009).* Chantilly, Virginia, USA: OWLED Steering Committee, 2009

[Gri14] Gringard, M.: *Erweiterung der lehrstuhleigenen Kontaktbibliothek um rotationssymmetrische Körper*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Diplomarbeit, 2014

[Gru93] Gruber, T. R.: A translation approach to portable ontology specifications. In: *Knowledge Acquisition* 5 (1993), S. 199–220

[GSA+11] Gausemeier, J.; Schäfer, W.; Anacker, H.; Bauer, F.; Dziwok, S.: Einsatz semantischer Technologien im Entwurf mechatronischer Systeme. In: *8. Paderborner Workshop Entwurf mechatronischer Systeme*, 2011, S. 7–35

[GSG+09] Gausemeier, J.; Schäfer, W.; Greenyer, J.; Kahl, S.; Pook, S.; Rieke, J.: Management of Cross-Domain Model Consistency During the Developement of Advanced Mechatronic Systems. In: *Proceeding of the International Conference on Engineering Design 2009 (ICED'09) - Section: Design Methods and Tools* Bd. 6. Palo Alto, CA, USA: The Design Society, August 2009, S. 1–12

[GTS14] Gausemeier, J. (Hrsg.); Trächtler, A. (Hrsg.); Schäfer, W. (Hrsg.): *Semantische Technologien im Entwurf mechatronischer Systeme: Effektiver Austausch von Lösungswissen in Branchenwertschöpfungsketten.* München: Carl Hanser Verlag, 2014. – Autoren: Anacker, H.; Bauer, F.; Borcherding, H.; Dziwok, S.; Frank, U.; Gausemeier, J.; Herden, R.; Hoppe, G.; Just, V.; Kiele-Dunsche, M.; Kruse, D.; Oestersötebier, F.; Papenfort, J.; Pohlmann, U.; Reddehase, H.; Rieke, J.; Schäfer, W.; Schierbaum, T.; Seifert, L.; Stichweh, H.; Teichrieb, H.; Trächtler, A.; Wagner, R.; Wessels, S.

[Gua98] Guarino, N.: Formal Ontology in Information Systems. In: *Proceedings of FOIS'98 Trento, Italien.* Amsterdam: IOS Press, 1998, S. 3–15

[Har10] Harchenko, J.: *Mechatronischer Entwurf eines neuartigen aktiven Fahrzeugfederungssystems für PKW unter Verwendung einer reversierbaren Flügelzellenpumpe*, Universität Paderborn, Dissertation, 2010

[HB11] Heath, T.; Bizer, C.: Linked data: Evolving the web into a global data space. In: *Synthesis lectures on the semantic web: theory and technology* 1 (2011), Nr. 1, S. 1–136

[HC75] Hunt, K. H.; Crossley, F. R. E.: Coefficient of Restitution Interpreted as Damping in Vibroimpact. In: *Journal of Applied Mechanics* 42 (1975), Juni, Nr. 2

[HdF+12] Haberfellner, R. (Hrsg.); de Weck, O. (Hrsg.); Fricke, E. (Hrsg.); Vössner, S. (Hrsg.): *Systems Engineering: Grundlagen und Anwendung*. Orell Füssli, 2012

[Hen12] Henderson-Sellers, B.: *On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages*. 2012 (SpringerBriefs in Computer Science)

[Hep06] Hepp, M.: Products and services ontologies: a methodology for deriving OWL ontologies from industrial categorization standards. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 2 (2006), Nr. 1, S. 72–99

[Hep07] Hepp, M.: Possible ontologies: How reality constrains the development of relevant ontologies. In: *Internet Computing, IEEE* 11 (2007), Nr. 1, S. 90–96

[HFK+11] Haskins, C. (Hrsg.); Forsberg, K. (Hrsg.); Krueger, M. (Hrsg.); Walden, D. (Hrsg.); Hamelin, R. D. (Hrsg.): *Systems Engineering Handbook – A Guide for System life cycle processes and activities v. 3.2.2*. International Council on Systems Engineering (INCOSE), 2011

[Hip04] Hippmann, G.: *Modellierung von Kontakten komplex geformter Körper in der Mehrkörperdynamik*, Technische Universität Wien, Dissertation, 2004

[HKH+14] Hodgson, R.; Keller, P. J.; Hodges, J.; Spivak, J.: *QUDT – Quantities, Units, Dimensions and Data Types in OWL and XML – Version 1.1*. `http://www.qudt.org/`. Version: 2014

[HKP+12] Hitzler, P.; Krötzsch, M.; Parsia, B.; Patel-Schneider, P. F.; Rudolph, S.: *OWL 2 Web Ontology Language - Primer (Second Edition)*. W3C. `http://www.w3.org/TR/owl2-primer/`. Version: Februar 2012

[HKR+08] Hitzler, P.; Krötzsch, M.; Rudolph, S.; Sure, Y.: *Semantic Web: Grundlagen*. Berlin: Springer, 2008

[Hor13] Horrocks, I.: What are ontologies good for? In: Küppers, B.-O. (Hrsg.); Hahn, U. (Hrsg.); Artmann, S. (Hrsg.): *Evolution of Semantic Systems*. Springer, 2013, S. 175–188

[HP12] Horridge, M.; Patel-Schneider, P. F.: *OWL 2 Web Ontology Language Manchester Syntax (Second Edition)*. `http://www.w3.org/TR/owl2-manchester-syntax/`. Version: Dezember 2012. – W3C Working Group Note

[HPB+04] Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M.: *SWRL: A semantic web rule language combining OWL and RuleML.* W3C Submission. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`. Version: Mai 2004

[HS13] Harris, S.; Seaborne, A.: *SPARQL 1.1 Query Language, W3C Recommendation.* `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`. Version: März 2013

[HSN+97] Honekamp, U.; Stolpe, R.; Naumann, R.; Lückel, J.: Structuring Approach for Complex Mechatronic Systems. In: *Proceedings of the 30th International Symposium on Automotive Technology & Automation (ISATA) – Mechatronics/Automotive Electronics.* Florenz, Italien: Automotive Automation Ltd., Juni 1997, S. 165–172

[HT96] Harashima, F.; Tomizuka, M.: Mechatronics - What Is It, Why, and How? In: *IEEE/ASME Transactions on Mechatronics* 1 (1996), Nr. 1

[IFA05] IFAC Technical Committee on Mechatronic Systems: *Scope.* Online-Quelle. `http://tc.ifac-control.org/4/2/scope`. Version: September 2005

[IKD+13] Iwanek, P.; Kaiser, L.; Dumitrescu, R.; Nyssen, A.: Fachdisziplinübergreifende Systemmodellierung mechatronischer Systeme mit SysML und CONSENS. In: Maurer, M. (Hrsg.); Schulze, S.-O. (Hrsg.): *Tag des Systems Engineerings*, Carl Hanser Verlag, November 2013, S. 337–346

[Ill14] Illg, I.: *Mechatronischer Entwurf und Erprobung einer regelbaren Federung für ein leichtes geländegängiges Kettenfahrzeug*, Universität Paderborn, Dissertation, 2014

[Ise08] Isermann, R.: *Mechatronische Systeme – Grundlagen.* 2., vollst. neu bearb. Aufl. Springer, 2008

[ISO01] Norm ISO 13584-1:2001 2001. *Industrial automation systems and integration – Parts library – Part 1: Overview and fundamental principles*

[ISO05] Norm ISO/IEC 19501:2005 2005. *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

[ISO11] Norm ISO/IEC 9075-1:2011 2011. *Information technology — Database languages — SQL*

[Jan09] Janschek, K.: *Systementwurf mechatronischer Systeme: Methoden – Modelle – Konzepte.* Berlin, Heidelberg: Springer, 2009

[JPB08] Johnson, T.; Paredis, C. J. J.; Burkhart, R.: Integrating Models and Simulations of Continuous Dynamics into SysML. In: *Proceedings of the 6th International Modelica Conference*. Bielefeld: Modelica Association, March 2008, S. 135–145

[JS13] Johnson, D.; Speicher, S.: *Open Services for Lifecycle Collaboration Core Specification Version 2.0*. `http://open-services.net/bin/view/Main/OslcCoreSpecification`. Version: Mai 2013

[Juh08] Juhász, T.: *Advanced Solutions in Object-Oriented Mechatronic Simulation*, Dept. of Control Engineering and Information Technology, Budapest University of Technology and Economics, Dissertation, 2008

[Jum15] Jumar, U.: Entwurf komplexer Automatisierungssysteme. In: *at - Automatisierungstechnik* 63 (2015), Nr. 2, S. 71–73

[Jus14a] Just, V.: *Modellbasierte Analyse und regelungstechnische Maßnahmen zur Optimierung des Vereinzelungsvorgangs in Bankautomaten*, Universität Paderborn, Dissertation, 2014

[Jus14b] Just, V.: *Modellbasierter Entwurf mechatronischer Systeme (Vorlesungsunterlagen WS 2014/2015)*. Universität Paderborn, 2014

[KBA12] Kraftfahrt-Bundesamt (Hrsg.): *Jahresbericht 2012*. 2012

[KBF+09] Krima, S. I.; Barbau, R. R.; Fiorentini, X.; Rachuri, R.; Sriram, R. D.: *OntoSTEP: OWL-DL Ontology for STEP – NIST Interagency/Internal Report (NISTIR) 7561*. 2009

[KFN+04] Knublauch, H.; Fergerson, R. W.; Noy, N. F.; Musen, M. A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: McIlraith, S. A. (Hrsg.); Plexousakis, D. (Hrsg.); Harmelen, F. van (Hrsg.): *The Semantic Web – ISWC 2004 Third International Semantic Web Conference* Bd. 3298. Springer Berlin Heidelberg, 2004, S. 229–243

[KGK03] Kosinska, A.; Galicki, M.; Kedzior, K.: Designing and Optimization of Parameters of Delta-4 Parallel Manipulator for a Given Workspace. In: *Journal of Robotic Systems* 20 (2003), Nr. 9

[KHI11] Knublauch, H.; Hendler, J. A.; Idehen, K.: *SPIN - Overview and Motivation*. W3C Member Submission. `http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/`. Version: Februar 2011

[Kie07] Kiel, E. (Hrsg.): *Antriebslösungen – Mechatronik für Produktion und Logistik*. Springer Berlin Heidelberg New York, 2007

[Kit06] Kitamura, Y.: Roles of ontologies of engineering artifacts for design knowledge modeling. In: *Proceedings of the 5th International Seminar and Workshop Engineering Design in Integrated Product Development (EDIProD 2006)*. Gronów, Polen, 2006, S. 59–69

[KK08] Kastens, U.; Kleine Büning, H.: *Modellierung – Grundlagen und formale Methoden*. München: Carl Hanser Verlag, 2008

[KK13] Kleiner, S.; Kramer, C.: Model Based Design with Systems Engineering Based on RFLP Using V6. In: Abramovici, M. (Hrsg.); Stark, R. (Hrsg.): *Smart Product Engineering*. Springer Berlin Heidelberg, 2013 (Lecture Notes in Production Engineering), S. 93–102

[KM03] Kitamura, Y.; Mizoguchi, R.: Ontology-based description of functional design knowledge and its use in a functional way server. In: *Expert Systems with Applications* 24 (2003), Nr. 2, S. 153–166

[KMS15a] Kaul, T.; Meyer, T.; Sextro, W.: Integrated Model for Dynamics and Reliability of Intelligent Mechatronic Systems. In: *Proceedings of the 25th annual European Safety and Reliability Conference ESREL*. Zürich, Schweiz: European Safety and Reliability Association, September 2015. – noch nicht veröffentlicht

[KMS15b] Kaul, T.; Meyer, T.; Sextro, W.: Integrierte Modellierung der Dynamik und der Verlässlichkeit komplexer mechatronischer Systeme. In: *10. Paderborner Workshop Entwurf mechatronischer Systeme*. Paderborn: HNI-Verlag, 2015, S. 101–112

[Koc05] Koch, T.: *Integration von Konstruktion und mechatronischer Komposition während des Entwurfs mechatronischer Systeme am Beispiel eines integrierten Radmoduls*, Universität Paderborn, Dissertation, 2005

[Koh96] Kohorst, H.: *Bevölkerungsentwicklung in China, Informations- und Kommunikationstechnologische Grundbildung in Nordrhein-Westfalen – Modellbildung und Simulation*. `http://www.kohorst-lemgo.de/modell/bevchina/china1.htm`. Version: 1996

[Kru] Kruse, D.: *Methodik zur teilautomatisierten Parameteridentifikation im Entwurf mechatronischer Systeme*, Universität Paderborn, Dissertation. – noch nicht veröffentlicht

[KS01] Kluge, F.; Seebold, E.: *Etymologisches Wörterbuch der deutschen Sprache*. 24. Walter de Gruyter, 2001

[KTT+06] Kawai, H.; Tanaka, F.; Takahashi, H.; Hashimoto, N.; Yamauchi, H.: Relationship Between Physical Properties of Dough

and Expansion Ability During Bread-Making. In: *Food Science and Technology Research* 12 (2006)

[Kuf11] Kufner, A.: *Automatisierte Erstellung von Maschinenmodellen für die Hardware-in-the-Loop-Simulation von Montagemaschinen*, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart, Dissertation, 2011

[KWH+11] Karban, R.; Weilkiens, T.; Hauber, R.; Zamparelli, M.; Diekmann, R.; Hein, A.: *Cookbook for MBSE with SysML*. `http://mbse.gfse.de/documents/SE2PracticesAndGuidelines.pdf`. Version: 2011

[Lan15] Langenscheidt GmbH & Co. KG (Hrsg.): *Langenscheidt Fremdwörterbuch: Methodik*. `http://de.langenscheidt.com/fremdwoerterbuch/Methodik`. Version: 2015. – abgerufen am 11.07.2015

[LG00] Lossack, R.; Grabowski, H.: The Axiomatic Approach in the Universal Design Theory. In: *Proceedings of ICAD2000 First International Conference on Axiomatic Design*. Cambridge: Institute for Axiomatic Design, Juni 2000, S. 203–210

[Lin09] Lindemann, U.: *Methodische Entwicklung technischer Produkte – Methoden flexibel und situationsgerecht anwenden*. Springer Verlag, 2009

[Lit79] Litz, L.: Ordnungsreduktion linearer Zustandsraummodelle durch Beibehaltung der dominanten Eigenbewegungen. In: *Regelungstechnik* 27 (1979), S. 80–86

[LKS00] Lückel, J.; Koch, T.; Schmitz, J.: Mechatronik als integrative Basis für innovative Produkte. In: VDI-Gesellschaft Entwicklung, Produktion, Vertrieb (Hrsg.): *Mechatronik – Mechanisch/Elektrische Antriebstechnik*. VDI-Verlag, 2000, S. 1–26

[LN94] Lankarani, H. M.; Nikravesh, P. E.: Continuous Contact Force Models for Impact Analysis in Multibody Systems. In: *Nonlinear Dynamics* 5 (1994)

[Loc] Lochbichler, M.: *Systematische Wahl einer Modellierungstiefe im Entwurfsprozess mechatronischer Systeme*, Universität Paderborn, Dissertation. – noch nicht veröffentlicht

[LOT14] Lochbichler, M.; Oestersötebier, F.; Trächtler, A.: Dynamic Behavior Models and their Modeling Depth in the Design Process of Mechatronic Systems. In: *Proceedings of the ASME 2014*

International Mechanical Engineering Congress & Exposition IMECE 2014. Montreal, Québec, Kanada: ASME, November 2014, S. V011T14A051

[LP97] Lobão, E.; Porto, A.: Proposta Para Sistematização de Estudos de Simulação (A simulation study systematization). In: *Proceedings of the XVII ENEGEP National Congress Of Industrial Engineering*. Gramado, Rio Grande do Sul, Brasilien: Brazilian Association of Industrial Engineering, 1997

[LP03] Liang, V.-C.; Paredis, C. J.: A Port Ontology for Automated Model Composition. In: Chick, S. (Hrsg.); Sanchez, P. J. (Hrsg.); Ferin, D. (Hrsg.); Morrice, D. J. (Hrsg.): *Proceedings of the 2003 Winter Simulation Conference*, 2003, S. 613–622

[LS99] Lassila, O.; Swick, R. R.: *Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation*. W3C. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222`. Version: Februar 1999

[LSB+12] Lochbichler, M.; Schmüdderrich, T.; Brökelmann, J.; Trächtler, A.: Methodology For Selecting The Modeling Depth Of Object-Oriented Behavioral Models. In: *International Conference on Modeling and Simulation*. Zürich: World Academy of Science, Engineering and Technology, Juli 2012, S. 327–331

[LW10] Lamm, J. G.; Weilkiens, T.: Funktionale Architekturen in SysML. In: Maurer, S.-O. M.; S. M.; Schulze (Hrsg.): *Tag des Systems Engineering 2010*. München: Carl Hanser Verlag, 2010, S. 109–118

[MAH+11] Mallea, A.; Arenas, M.; Hogan, A.; Polleres, A.: On blank nodes. In: *The Semantic Web–ISWC 2011*. Springer, 2011, S. 421–437

[MDS+05] Makkar, C.; Dixon, W. E.; Sawyer, W. G.; Hu, G.: A New Continuously Differentiable Friction Model for Control Systems Design. In: *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. Monterey, CA, USA: IEEE, 2005, S. 600–605

[Mey10] Meyna, A.: Zuverlässigkeit elektronischer Systeme im Kraftfahrzeug. In: *BUW.Output - Forschungsmagazin der Bergischen Universität Wuppertal* (2010), Nr. 3

[Mez10] Mezger, T.: *Das Rheologie Handbuch: Für Anwender von Rotations- und Oszillations-Rheometern*. 3. Vincentz Network GmbH & Company KG, 2010

[MK01] Mizoguchi, R.; Kitamura, Y.: Foundation of Knowledge Systematization: Role of Ontological Engineering. In: Roy, R. (Hrsg.): *Industrial Knowledge Management*. Springer London, 2001, S. 17–36

[MMP+04] Mocko, G.; Malak, R. J.; Paredis, C. J.; Peak, R.: A Knowledge Repository for Behavioral Models in Engineering Design. In: *ASME Design Engineering Technical Conf., Computers and Information in Engineering Conf.* American Society of Mechanical Engineers (ASME), 2004, S. 943–952

[Mod12] Modelica Association: *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling – Language Specification, Version 3.3*. `http://www.modelica.org`. Version: Mai 2012

[Mod13] Modelica Association: *Modelica User's Guide*. August 2013

[Mot07] Motik, B.: On the properties of metamodeling in OWL. In: *Journal of Logic and Computation* 17 (2007), Nr. 4, S. 617–637

[Neu90] Neunzert, H.: Von Modellen und wie man sie nutzt. In: *Forum '90 Wissenschaft und Technik, Neue Anwendungen mit Hilfe aktueller Computer-Technologien*. Trier, Oktober 1990, S. 10–21

[Nou77] Nour Eldin, H. A.: Minimalrealisierung der Matrix-Übertragungsfunktion/Minimal realization of the matrix transfer function. In: *at-Automatisierungstechnik* 25 (1977), Nr. 1-12, S. 82–87

[NT97] Nonaka, I.; Takeuchi, H.: *Die Organisation des Wissens – Wie japanische Unternehmen eine brachliegende Ressource nutzbar machen*. Frankfurt am Main: Campus Verlag, 1997

[OBM02] Otto, B.; Beckmann, H.; Müller, S.: E-Business-Standards: Verbreitung und Akzeptanz. In: Fraunhofer IAO, S. (Hrsg.): *media vision special*. Fraunhofer IRB Verlag, 2002, S. 1–89

[OD09] O'Connor, M. J.; Das, A. K.: SQWRL: a query language for OWL. In: *OWL: Experiences and Directions (OWLED), Sixth International Workshop*. Chantilly, Virginia, USA: OWLED Steering Committee, 2009

[ODB+12] Oestersötebier, F.; Dziwok, S.; Bauer, F.; Trächtler, A.; Schäfer, W.; Gausemeier, J.: Unterstützung des mechatronischen Entwurfs durch die effektive Suche nach Lösungselementen mithilfe von semantischen Technologien. In: *Tag des Systems Engineering*, 2012, S. 207–216

[OEL05] Otter, M.; Elmqvist, H.; López, J. D.: Collision Handling for the Modelica MultiBody Library. In: *Proceedings of the 4th International Modelica Conference*, 2005, S. 45–53

[OJT+12] Oestersötebier, F.; Just, V.; Trächtler, A.; Bauer, F.; Dziwok, S.: Model-based Design of Mechatronic Systems by Means of Semantic Web Ontologies and Reusable Solution Elements. In: *Proceedings of the ASME 2012 11th Biennial Conference On Engineering Systems Design And Analysis (ESDA2012)* ASME, 2012, S. 647–656

[OMG07] Spezifikation Object Management Group November 2007. *OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*. `http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF`

[OMG12] Spezifikation Object Management Group Juni 2012. *OMG Systems Modeling Language – Version 1.3*. `http://www.omg.org/spec/SysML/1.3/`

[ONB02] Overstreet, C.; Nance, R. E.; Balci, O.: Issues in enhancing model reuse. In: *Proceedings of the International Conference on Grand Challenges for Modeling and Simulation*, 2002, S. 27–31

[OWT14] Oestersötebier, F.; Wang, P.; Trächtler, A.: A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces. In: *10th International Modelica Conference*. Lund, Sweden: Modelica Association, März 2014, S. 929–937

[Pan88] Pantelides, C. C.: The Consistent Initialization of Differential-Algebraic Systems. In: *SIAM Journal on Scientific and Statistical Computing* 9 (1988), Nr. 2, S. 213–231

[PBB+10] Paredis, C.; Bernard, Y.; Burkhart, R. M.; Koning, H.-P. de; Friedenthal, S.; Fritzson, P.; Rouquette, N. F.; Schamai, W.: An overview of the SysML-Modelica transformation specification. In: *INCOSE International Symposium*, 2010, S. 11–15

[PBF+13] Pahl, G. (Hrsg.); Beitz, W. (Hrsg.); Feldhusen, J. (Hrsg.); Grote, K.-H. (Hrsg.): *Pahl/Beitz Konstruktionslehre: Methoden und Anwendung erfolgreicher Produktentwicklung*. 8. Berlin: Springer Verlag, 2013

[PC85] Parnas, D. L.; Clements, P. C.: A Rational Design Process: How and Why to Fake It. In: Ehrig, H. (Hrsg.); Floyd, C. (Hrsg.); Nivat, M. (Hrsg.); Thatcher, J. W. (Hrsg.): *Mathematical Foundations of Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Berlin, Germany, March 25–29, 1985, Volume 2: Colloquium*

on Software Engineering (CSE) Bd. 186, Springer, 1985 (Lecture Notes in Computer Science), S. 80–100

[PDS+01] Paredis, C.; Diaz-Calderon, A.; Sinha, R.; Khosla, P.: Composable Models for Simulation-Based Design. In: *Engineering with Computers* 17 (2001), S. 112–128

[PDS05] Patil, L.; Dutta, D.; Sriram, R.: Ontology-based exchange of product data semantics. In: *IEEE Transactions on Automation Science and Engineering* 2 (2005), Nr. 3, S. 213–225

[PF03] Pop, A.; Fritzson, P.: ModelicaXML: A Modelica XML Representation with Applications. In: *3rd International Modelica Conference*. Linköping, Sweden: Modelica Association, 2003

[PF05] Park, M.; Fishwick, P. A.: Integrating Dynamic and Geometry Model Components through Ontology-Based Inference. In: *SIMULATION* 81 (2005), Nr. 12, S. 795–813

[Pfe12] Pfeiffer, A.: Optimization Library for Interactive Multi-Criteria Optimization Tasks. In: *Proceedings of the 9th International Modelica Conference*. München: Modelica Association, September 2012, S. 669–680

[Pic03] Pickert, G.: *Einführung in Ontologien*, Humboldt-Universität zu Berlin, Seminararbeit, Seminar Grundlegende Aspekte des Semantic Web (Wintersemester 2002/03), 2003. `http://www.dbis.informatik.hu-berlin.de/dbisold/lehre/WS0203/SemWeb/artikel/2/Pickert_Ontologien_final.pdf`

[Pop10] Popov, V. L.: *Kontaktmechanik und Reibung – Von der Nanotribologie bis zur Erdbebendynamik*. 2. Springer, 2010

[PPC+10] Plateaux, R.; Penas, O.; Choley, J.-Y.; M'henni, F.; Rivière, A.: Integrated design methodology of a mechatronic system. In: *Mécanique & Industries* 11 (2010), Nr. 5, S. 401–406

[Pra12] Prahl, N.: *Ermittlung der Inversen Kinematik für den Demonstrator im Forschungsprojekt ENTIME*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Bachelorarbeit, 2012. – betreut durch Heinrich Teichrieb

[QP12] Qamar, A.; Paredis, C. J. J.: Dependency Modeling and Model Management in Mechatronic Design. In: *Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. Chicago, IL, USA: ASME, August 2012

[RAC12] Racer Systems GmbH & Co. KG (Hrsg.): *RacerPro User's Guide Version 2.0.* `http://racer.sts.tuhh.de/Racer-2.0/users-guide-2-0.pdf`. Version: Oktober 2012

[Rau08] Rautenberg, W.: *Einführung in die Mathematische Logik - Ein Lehrbuch.* 3. Vieweg und Teubner Verlag, 2008

[RH06] Ruiz, F.; Hilera, J. R.: Using Ontologies in Software Engineering and Technology. In: Calero, C. (Hrsg.); Ruiz, F. (Hrsg.); Piattini, M. (Hrsg.): *Ontologies for Software Engineering and Software Technology.* Springer-Verlag Berlin Heidelberg, 2006, S. 49–102

[Rie14] Rieke, J.: *Model Consistency Management for Systems Engineering*, Universität Paderborn, Dissertation, 2014

[RNP+04] Robinson, S.; Nance, R. E.; Paul, R. J.; Pidd, M.; Taylor, S. J.: Simulation model reuse: definitions, benefits and obstacles. In: *Simulation modelling practice and theory* 12 (2004), Nr. 7, S. 479–494

[Rol95] Roller, D.: *CAD - effiziente Anpassungs- und Variantenkonstruktion.* Springer Verlag, 1995

[Rot01] Roth, K.: *Konstruieren mit Konstruktionskatalogen: Band 2 – Kataloge.* Berlin: 3. Auflage, Springer Verlag, 2001

[RP11] Reichwein, A.; Paredis, C. J.: Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering. In: *Proceedings of the ASME 2011 International Design Engineering Technical Conference & Computers and Information in Engineering Conference*, 2011, S. 1341–1349

[RPC+12] Reichwein, A.; Paredis, C. J. J.; Canedo, A.; Witschel, P.; Stelzig, P. E.; Votintseva, A.; Wasgint, R.: Maintaining Consistency Between System Architecture and Dynamic System Models with SysML4Modelica. In: *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling.* New York, NY, USA: ACM, 2012 (MPM '12), S. 43–48

[RRC06] Rodrigues, T.; Rosa, P.; Cardoso, J.: Mapping XML to Exiting OWL ontologies. In: *International Conference WWW/Internet* Citeseer, 2006, S. 72–77

[SA11] Subhashini, R.; Akilandeswari, J.: A survey on ontology construction methodologies. In: *International Journal of Enterprise Computing and Business Systems* 1 (2011), Nr. 1, S. 60–72

[San00] Sander, S.: *Konzept einer digitalen Lösungsbibliothek für die integrierte Produktentwicklung*, Technische Fakultät der Universität Erlangen-Nürnberg, Dissertation, 2000

[SBF98] Studer, R.; Benjamin, V. R.; Fensel, D.: Knowledge engineering: Principles and methods. In: *Data & Knowledge Engineering* 25 (1998), Nr. 1–2, S. 161–197

[SCD+12] Shehzad, A.; Chiron, H.; Della Valle, G.; Lamrini, B.; Lourdin, D.: Energetical and rheological approaches of wheat flour dough mixing with a spiral mixer. In: *Journal of Food Engineering* 110 (2012), S. 60–70

[Sch94] Schürr, A.: Specification of graph translators with Triple Graph Grammars. In: Mayr, E. W. (Hrsg.); Schmidt, G. (Hrsg.); Tinhofer, G. (Hrsg.): *20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 94)*, Springer Verlag, 1994, S. 151–163

[Sch12a] Schmiegel, B.: *Idealisierte Modellierung elektromagnetischer Linear-/Planarmotoren in Dymola/Modelica*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Bachelorarbeit, 2012

[Sch12b] Schulte, F.: *Idealisierte Modellierung von Sensoren in Dymola/Modelica*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Bachelorarbeit, 2012

[Sch15] Schaeffler Technologies AG & Co. KG (Hrsg.): *Wälzlager – Reibung und Erwärmung.* `http://medias.schaeffler.com/medias/de!hp.tg.cat/tg_hr*ST4_102160011;a9dJnuipBB76`. Version: Januar 2015

[SGM08] Stelzmann, U.; Groth, C.; Müller, G.: *FEM für Praktiker - Band 2: Strukturdynamik.* 5. Expert-Verlag, 2008

[SHJ+06] Schmitz, C.; Hotho, A.; Jäschke, R.; Stumme, G.: Kollaboratives Wissensmanagement. In: Pellegrini, T. (Hrsg.); Blumauer, A. (Hrsg.): *Semantic Web - Wege zur vernetzten Wissensgesellschaft.* Springer-Verlag Berlin Heidelberg, 2006, S. 273–289

[Sil10] Silveira Mastella, L.: *Semantic exploitation of engineering models: application to petroleum reservoir models*, École nationale supérieure des mines de Paris, Dissertation, 2010

[SKS+11] Schmutzler, C.; Krüger, A.; Simons, M.; Becker, J.: Ansätze zur Integration von energieeffizienten Intelligenten Kommunikationskontrollern für FlexRay in Autosar. In: *Informatik 2011-41. Jahrestagung der Gesellschaft für Informatik*, 2011

[SLB+13] Schmüdderrich, T.; Lochbichler, M.; Brökelmann, J.; Trächtler, A.: Methodik zur anforderungsgerechten Wahl der

Modellierungstiefe von Verhaltensmodellen für die virtuelle Inbetriebnahme. In: *Tagungsband Mechatronik 2013*, 2013, S. 43–48

[SN12] Saxena, P.; Nosrati, M.: Review of Slug Semantic Web. In: *World Applied Programming* 2 (2012), Januar, Nr. 1

[SPC+07] Sirin, E.; Parsia, B.; Cuenca Grau, B.; Kalyanpur, A.; Katz, Y.: Pellet: A Practical OWL-DL Reasoner. In: *Journal of Web Semantics* 5 (2007), Nr. 2. `http://pellet.owldl.com/papers/sirin07pellet.pdf`

[SR14] Schreiber, G.; Raimond, Y.: *RDF 1.1 Primer*. W3C. `http://www.w3.org/TR/2014/NOTE-rdf11--20140624/`. Version: Februar 2014

[SS06] Sahin, S.; Sumnu, S. G.: *Physical Properties of Foods*. New York: Springer Science and Business Media, LLC, 2006

[ST11] Salminen, A.; Tompa, F.: *Communicating with XML*. New York, Dordrecht, Heidelberg, London: Springer, 2011

[Sta73] Stachowiak, H.: *Allgemeine Modelltheorie*. Springer-Verlag, 1973

[Str96] Strahringer, S.: *Metamodellierung als Instrument des Methodenvergleichs: eine Evaluierung am Beispiel objektorientierter Analysemethoden*. Shaker Verlag, 1996 (Berichte aus der Betriebswirtschaft)

[Str98] Strahringer, S.: Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips. In: Pohl, K. (Hrsg.); Schürr, A. (Hrsg.); Vossen, G. (Hrsg.): *Modellierung* Bd. 9, CEUR-WS.org, 1998 (CEUR Workshop Proceedings)

[Suh93] Suhm, A.: *Produktmodellierung in wissensbasierten Konstruktionssystemen auf Basis von Lösungsmustern*. Aachen, Fakultät für Maschinenbau, Universität Karlsruhe, Dissertation, 1993

[Sva11] Svaricek, F.: *Moderne Methoden der Regelungstechnik (Vorlesungsunterlagen WS 2011/2012)*. Universität der Bundeswehr München, 2011

[TH06] Tsarkov, D.; Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: *Proceedings of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)* Bd. 4130, Springer, 2006 (Lecture Notes in Artificial Intelligence), S. 292–297

[Tho15] Thommen, J.-P.; Springer Gabler Verlag (Hrsg.): *Gabler Wirtschaftslexikon, Stichwort: Anspruchsgruppen*. `http://wirtschaftslexikon.gabler.de/Archiv/1202/anspruchsgruppen-v6.html`. Version: Februar 2015

[Toe02] Toepper, S.: *Die mechatronische Entwicklung des Parallelroboters TRIPLANAR*, Universität Paderborn, Dissertation, 2002

[Tra13] Traphöner, P.: *Modellierung der Strömungsverhältnisse in einem Teigkneter*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Bachelorarbeit, 2013

[Tra14] Traphöner, P.: *Sensorauswertung zur Erkennung des Teigzustands in einem Teigkneter*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Studienarbeit, 2014

[TT14] Treue, I.; Teutsch, S.: Systems Engineering, aber wie? In: *Tag des Systems Engineering 2014 - Der Weg zu technischen Systemen von morgen.* Bremen: Gesellschaft für Systems Engineering e.V., 12.–14. November 2014, S. 103–111

[TUK96] Tomiyama, T.; Umeda, Y.; Kiriyama, T.: A framework for knowledge intensive engineering. In: *Computer Aided Systems Theory – CAST'94.* Springer, 1996, S. 123–147

[UNS14] GS1 US for the UN Development Programme (UNDP): *United Nations Standard Products and Services Code (UNSPSC) – Version 16.* 2014

[VDI04] Richtlinie VDI2206 Juni 2004. *Entwicklungsmethodik für mechatronische Systeme*

[Veh14] Vehmeier, P.: *Idealisierte Modellierung von Förderbändern in Dymola/Modelica*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Bachelorarbeit, 2014

[VWB+09] Vajna, S.; Weber, C.; Bley, H.; Zeman, K.: *CAx für Ingenieure – Eine praxisbezogene Einführung.* Springer Verlag, 2009

[Wag06] Wagner, C.: Breaking the Knowledge Acquisition Bottleneck Through Conversational Knowledge Management. In: *Information Resources Management Journal* 19 (2006), Nr. 1, S. 70–83

[Wal95] Wallaschek, J.: *Modellierung und Simulation als Beitrag zur Verkürzung der Entwicklungszeit mechatronischer Produkte.* VDI Berichte Nr. 1215, 1995

[Wan11] Wang, P.: *Idealisierte Modellierung von Kontakten in Dymola/Modelica*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Studienarbeit, 2011

[Wan12] Wang, P.: *Aufbau einer Kontaktbibliothek für einfache Kontaktgeometrien in Dymola/Modelica*, Universität Paderborn, Lehrstuhl für Regelungstechnik und Mechatronik, Masterarbeit, 2012

[Wei06] Weilkiens, T.: *Systems Engineering mit SysML/UML: Modellierung, Analyse, Design.* Heidelberg: dpunkt Verlag, 2006

[Wei07] Weilkiens, T.: *Systems engineering with SysML/UML: modeling, analysis, design.* Burlington, MA, USA: Morgan Kaufmann Publishers, 2007

[WIC+15] Weisske, A.; Iwanek, P.; Czaja, A.; Flottmeier, S.; Trächtler, A.; Dumitrescu, R.: Planung der Inbetriebnahme mechatronischer Prototypen auf Basis einer disziplinübergreifenden Systemmodellierung. In: *Tagungsband Mechatronik.* Dortmund: VDI Mechatronik, März 2015, S. 231–237

[Wit02] Wittler, G.: *Integrative Modellierung von Gestalt und dynamischem Verhalten beim Entwurf mechatronischer Systeme*, Universität Paderborn, Dissertation, 2002

[WWH08] Waterfeld, W.; Weiten, M.; Haase, P.: Ontology Management Infrastructures. In: Hepp, M. (Hrsg.); De Leenheer, P. (Hrsg.); De Moor, A. (Hrsg.); Sure, Y. (Hrsg.): *Ontology Management* Bd. 7. Springer US, 2008, S. 59–87

[YDM08] Yang, D.; Dong, M.; Miao, R.: Development of a product configuration system with an ontology-based approach. In: *Computer-Aided Design* 40 (2008), Nr. 8, S. 863–878

[Yos85] Yoshikawa, T.: Manipulability of Robotic Mechanisms. In: *The International Journal of Robotics Research* 4 (1985), Nr. 2

[Yos00] Yoshioka, M.: *Knowledge Intensive Engineering Framework: KIEF (formerly known as SYSFUND) – Manual.* `http://www-kb.ist.hokudai.ac.jp/~yoshioka/KIEF/manual.pdf`. Version: 2000

[YUT+04] Yoshioka, M.; Umeda, Y.; Takeda, H.; Shimomura, Y.; Nomaguchi, Y.; Tomiyama, T.: Physical concept ontology for the knowledge intensive engineering framework. In: *Advanced Engineering Informatics* 18 (2004), Nr. 2, S. 95–113

[Zha07] Zhan, P.: *An ontology-based approach for semantic level information exchange and integration in applications for product lifecycle management*, Washington State University, Dissertation, 2007

[ZW06] Zirn, O.; Weickert, S.: *Modellbildung und Simulation hochdynamischer Fertigungssysteme.* Springer-Verlag, Berlin, 2006

A Anhang

A.1 Weiterführendes Beispiel zur Bestimmung von Komplexität und Gesamtmodellierungstiefe

Die drei in Bild A-1 dargestellten Beispiele sollen weiterführende Überlegungen zur Bestimmung der Modellkomplexität K und der Gewichtungsfaktoren γ zur Berechnung der Gesamtmodellierungstiefe verdeutlichen (vgl. Abschnitt 4.1.2). Die Abbildung zeigt drei Möglichkeiten der Modellkopplung. Die identischen Teilmodelle bestehen in diesem akademischen Beispiel lediglich aus jeweils einer Drehträgheit J_1 und J_2. Die Anzahl der Minimalzustände beträgt in allen drei Fällen $n_{min,1} = n_{min,2} = 2$. Da die beiden Einzelmodelle dieselbe Modellierungstiefe $\Gamma_{1,2} = 2$ besitzen, resultiert diese auch für das Gesamtmodell. Wie verhält es sich jedoch mit der Modellkomplexität des Gesamtmodells und der γ-Faktoren? Zur Beantwortung dieser Frage sind die durch die Kopplung entstehenden Dynamikgleichungen des Gesamtmodells jeweils rechts aufgeführt.

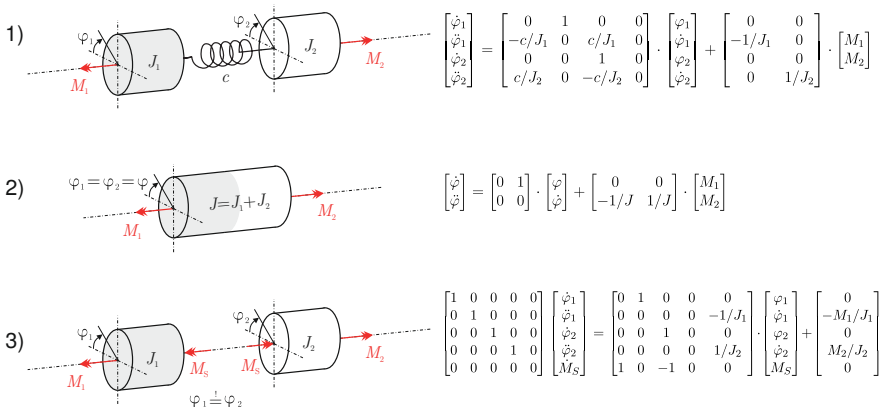


Bild A-1: Drei Beispiele zur Verdeutlichung verschiedener Möglichkeiten der Kopplung

In Beispiel 1 wird eine Elastizität mit der (sehr großen) Steifigkeit c als dynamische Kopplung eingefügt, um die Teilmodelle zu verbinden. Dadurch entsteht ein Gesamtmodell mit $n_{ges\,1} = \sum n = n_1 + n_2 = 4$ Zuständen. Beispiel 2 zeigt eine ideal starre Kopplung der Drehträgheiten, dadurch lassen sie sich zu der Gesamtträgheit $J = J_1 + J_2$ zusammenfassen und die Anzahl der Gesamtzustände reduziert sich auf $n_{ges\,2} = 2$. Da in beiden Fällen gewöhnliche Differentialgleichungen entstehen,

kann die Modellkomplexität analog zur Vorschrift 4-1 auf Seite 87 ermittelt werden. Mit $K_1 = 4$ ist sie im ersten Fall doppelt so groß wie in Beispiel 2. Die Größe *Komplexität* kann somit tatsächlich Aufschluss über den zu erwartenden Rechenaufwand geben, denn auch dieser ist bei der dynamischen Kopplung deutlich höher, insbesondere da die Steifigkeiten typischerweise sehr hoch sind.

Beispiel 3 nutzt eine kinematische Zwangsbedingung zur starren Kopplung von J_1 und J_2, was automatisch eine Differential-algebraische Gleichung zur Folge hat. Die Dimension des entstehenden Deskriptorsystems wird im Vergleich zum System im Zustandsraum (Beispiel 1) durch die algebraische Gleichung um eins größer. Gleichzeitig steigt auch der Rechenaufwand für Simulationswerkzeuge, denn diese müssen in der Regel Indexreduktionsalgorithmen[1](Differentiation der Zwangsbedingung nach der Zeit, siehe z. B. [Pan88]) anwenden und konsistente Anfangsbedingungen finden, um das Modell mit gängigen numerischen Lösungsverfahren berechnen zu können. Das hier in Beispiel 3 gezeigte System besitzt den Index 3, da die Zwangsbedingung dreimal nach der Zeit abgeleitet werden muss, um eine gewöhnliche DGL für das Zwangsmoment M_S zu erhalten (vgl. [Jan09, S. 142 ff.]). Durch diese Indexreduktion wird das Modell in ein ODE-System der Ordnung 5 überführt. Um den Begriff der Modellkomplexität auch auf Deskriptorsysteme anwenden zu können, wird daher vorgeschlagen, ihn in diesen Fällen mit der Dimension des Deskriptors bzw. der Ordnung des index-reduzierten Systems gleichzusetzen.

Im Gegensatz zur so bestimmten Modellkomplexität, zeigen sich die Unterschiede zwischen den Kopplungen nicht in der Gesamtmodellierungstiefe Γ_{ges}. Die Gewichtungsfaktoren der beiden Teilsysteme betragen jeweils $\gamma_{1,2} = (1+2)/(2+4) = 1/2$ (vgl. 4-3 auf Seite 92). Auch bei etwaigen Gewichtungsfaktoren, die bei einer Verknüpfung des Gesamtmodells mit anderen Modellen errechnet werden, ergeben sich keine Unterschiede. Denn die Anzahl der Minimalzustände beträgt in jedem Fall $n_{min,\,ges} = 2$. Beispiel 2 stellt demnach eine Minimalrealisierung dar.

A.2 ModelXML Schema

```
1  <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/
       XMLSchema">
2    <xs:element name="modelxml">
3      <xs:complexType>
4        <xs:choice maxOccurs="unbounded" minOccurs="0">
5          <xs:element name="package" maxOccurs="unbounded" minOccurs="0">
6            <xs:complexType>
7              <xs:sequence>
8                <xs:element type="xs:string" name="fullPath"/>
9                <xs:element type="xs:string" name="packageName"/>
10               <xs:element name="model" maxOccurs="unbounded" minOccurs="0">
```

[1] Der differentielle Index gibt die minimale Anzahl der nötigen zeitlichen Differentiation der algebraischen Zwangsbedingungen an, um ein System rein aus gewöhnlichen Differentialgleichungen zu erhalten [Jan09].

```
11                <xs:complexType>
12                  <xs:sequence>
13                    <xs:element type="xs:string" name="modelID"/>
14                    <xs:element type="xs:string" name="fullPath"/>
15                    <xs:element type="xs:string" name="modelType"/>
16                    <xs:element type="xs:string" name="modelName"/>
17                    <xs:element type="xs:string" name="extends" minOccurs="0"/>
18                    <xs:element type="xs:string" name="solver"/>
19                    <xs:element type="xs:string" name="modelLanguage"/>
20                  </xs:sequence>
21                </xs:complexType>
22              </xs:element>
23              <xs:element name="package" maxOccurs="unbounded" minOccurs="0">
24                <xs:complexType>
25                  <xs:sequence>
26                    <xs:element type="xs:string" name="fullPath"/>
27                    <xs:element type="xs:string" name="packageName"/>
28                    <xs:element name="model" minOccurs="0">
29                      <xs:complexType>
30                        <xs:sequence>
31                          <xs:element type="xs:string" name="modelID"/>
32                          <xs:element type="xs:string" name="fullPath"/>
33                          <xs:element type="xs:string" name="modelType"/>
34                          <xs:element type="xs:string" name="modelName"/>
35                          <xs:element type="xs:string" name="extends"/>
36                          <xs:element type="xs:string" name="solver"/>
37                          <xs:element name="component">
38                            <xs:complexType>
39                              <xs:sequence>
40                                <xs:element name="SystemElementModel">
41                                  <xs:complexType>
42                                    <xs:sequence>
43                                      <xs:element type="xs:string" name="SystemElementName"/>
44                                      <xs:element type="xs:string" name="SystemElementID"/>
45                                      <xs:element type="xs:string" name="baseModelID"/>
46                                      <xs:element type="xs:string" name="baseModelPath"/>
47                                      <xs:element name="modifier" maxOccurs="unbounded" minOccurs="0">
48                                        <xs:complexType>
49                                          <xs:sequence>
50                                            <xs:element type="xs:string" name="parameterName"/>
51                                            <xs:element type="xs:string" name="parameterID"/>
52                                            <xs:element type="xs:string" name="parameterValue"/>
53                                          </xs:sequence>
54                                        </xs:complexType>
55                                      </xs:element>
56                                    </xs:sequence>
57                                  </xs:complexType>
58                                </xs:element>
59                              </xs:sequence>
60                            </xs:complexType>
61                          </xs:element>
62                          <xs:element name="connection" maxOccurs="unbounded" minOccurs="0">
63                            <xs:complexType>
64                             <xs:sequence>
65                              <xs:element type="xs:string" name="connect" maxOccurs="unbounded" minOccurs="0"/>
66                             </xs:sequence>
67                            </xs:complexType>
68                          </xs:element>
69                          <xs:element type="xs:string" name="modelLanguage"/>
70                        </xs:sequence>
71                      </xs:complexType>
72                    </xs:element>
73                    <xs:element type="xs:string" name="SystemElementModel" minOccurs="0"/>
74                  </xs:sequence>
75                </xs:complexType>
76              </xs:element>
77            </xs:sequence>
78          </xs:complexType>
79        </xs:element>
80        <xs:element name="model">
81          <xs:complexType>
82            <xs:sequence>
83              <xs:element type="xs:string" name="modelID"/>
84              <xs:element type="xs:string" name="fullPath"/>
85              <xs:element type="xs:string" name="modelType"/>
86              <xs:element type="xs:string" name="modelName"/>
87              <xs:element type="xs:string" name="solver"/>
88              <xs:element name="component" maxOccurs="unbounded" minOccurs="0">
89                <xs:complexType>
90                  <xs:sequence>
91                    <xs:element name="interface">
92                      <xs:complexType>
93                        <xs:sequence>
94                          <xs:element type="xs:string" name="interfaceName"/>
```

```
95                          <xs:element type="xs:string" name="interfaceID"/>
96                          <xs:element type="xs:string" name="modelicaType"/>
97                          <xs:element type="xs:string" name="interfaceType"/>
98                        </xs:sequence>
99                      </xs:complexType>
100                   </xs:element>
101                 </xs:sequence>
102               </xs:complexType>
103             </xs:element>
104             <xs:element type="xs:string" name="modelLanguage"/>
105           </xs:sequence>
106         </xs:complexType>
107       </xs:element>
108       <xs:element name="InterfaceModel">
109         <xs:complexType mixed="true">
110           <xs:sequence>
111             <xs:element type="xs:string" name="modelID" minOccurs="0"/>
112             <xs:element type="xs:string" name="fullPath" minOccurs="0"/>
113             <xs:element type="xs:string" name="modelType" minOccurs="0"/>
114             <xs:element type="xs:string" name="modelName" minOccurs="0"/>
115             <xs:element type="xs:string" name="solver" minOccurs="0"/>
116             <xs:element name="component" maxOccurs="unbounded" minOccurs="0">
117               <xs:complexType>
118                 <xs:sequence>
119                   <xs:element name="interface">
120                     <xs:complexType>
121                       <xs:sequence>
122                         <xs:element type="xs:string" name="interfaceName"/>
123                         <xs:element type="xs:string" name="interfaceID"/>
124                         <xs:element type="xs:string" name="modelicaType"/>
125                         <xs:element type="xs:string" name="interfaceType"/>
126                       </xs:sequence>
127                     </xs:complexType>
128                   </xs:element>
129                 </xs:sequence>
130               </xs:complexType>
131             </xs:element>
132             <xs:element type="xs:string" name="modelLanguage" minOccurs="0"/>
133           </xs:sequence>
134         </xs:complexType>
135       </xs:element>
136       <xs:element type="xs:string" name="IntegratedSystemDynamicsModel"/>
137       <xs:element type="xs:string" name="SystemElementModel" maxOccurs="unbounded" minOccurs="0"/>
138       <xs:element type="xs:string" name="EnvironmentElementModel"/>
139       <xs:element name="SolutionPatternModel">
140         <xs:complexType mixed="true">
141           <xs:sequence>
142             <xs:element type="xs:string" name="modelID" minOccurs="0"/>
143             <xs:element type="xs:string" name="fullPath" minOccurs="0"/>
144             <xs:element type="xs:string" name="modelType" minOccurs="0"/>
145             <xs:element type="xs:string" name="modelName" minOccurs="0"/>
146             <xs:element type="xs:string" name="solver" minOccurs="0"/>
147             <xs:element name="component" maxOccurs="unbounded" minOccurs="0">
148               <xs:complexType>
149                 <xs:sequence>
150                   <xs:element name="interface">
151                     <xs:complexType>
152                       <xs:sequence>
153                         <xs:element type="xs:string" name="interfaceName"/>
154                         <xs:element type="xs:string" name="interfaceID"/>
155                         <xs:element type="xs:string" name="modelicaType"/>
156                         <xs:element type="xs:string" name="interfaceType"/>
157                       </xs:sequence>
158                     </xs:complexType>
159                   </xs:element>
160                 </xs:sequence>
161               </xs:complexType>
162             </xs:element>
163             <xs:element type="xs:string" name="annotation" minOccurs="0"/>
164             <xs:element name="connection" maxOccurs="unbounded" minOccurs="0">
165               <xs:complexType>
166                 <xs:sequence>
167                   <xs:element type="xs:string" name="connect" maxOccurs="unbounded" minOccurs="0"/>
168                 </xs:sequence>
169               </xs:complexType>
170             </xs:element>
171             <xs:element type="xs:string" name="modelLanguage" minOccurs="0"/>
172           </xs:sequence>
173         </xs:complexType>
174       </xs:element>
175     </xs:choice>
176   </xs:complexType>
177 </xs:element>
178 </xs:schema>
```

A.3 XSL-Code zur Transformation von ModelicaXML nach ModelXML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xslt="http://xml.apache.org/xslt">
 <xsl:output method="xml" indent="yes" xslt:indent-amount="4" />
 <xsl:template match="/">
  <xsl:element name="modelxml">
   <xsl:for-each select="/modelica/modelicaxml">
    <xsl:apply-templates select="definition[@restriction='package']">
     <xsl:with-param name="within">
      <xsl:value-of select="@within"/>
     </xsl:with-param>
    </xsl:apply-templates>
    <!-- search for models in case no package is present -->
    <xsl:apply-templates select="definition[@restriction='model']">
     <xsl:with-param name="within">
      <xsl:value-of select="@within"/>
     </xsl:with-param>
    </xsl:apply-templates>
   </xsl:for-each>
  </xsl:element>
 </xsl:template>

 <!-- template for packages -->
 <xsl:template match="definition[@restriction='package']">
  <xsl:param name="within" />
  <!-- create correct full path -->
  <xsl:variable name="temp_within">
   <xsl:choose>
    <xsl:when test="string-length($within) > 0">
     <xsl:value-of select="concat($within,'.',@ident)"/>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="@ident"/>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:variable>
  <!-- create package node -->
  <xsl:element name="{@restriction}">
   <xsl:element name="fullPath">
    <xsl:value-of select="$temp_within"/>
   </xsl:element>
   <xsl:element name="packageName">
    <xsl:value-of select="@ident"/>
   </xsl:element>
   <!-- search for models -->
   <xsl:apply-templates select="definition[@restriction='model']">
    <xsl:with-param name="within">
     <xsl:value-of select="$temp_within"/>
    </xsl:with-param>
   </xsl:apply-templates>
   <!-- search for other packages -->
   <xsl:apply-templates select="definition[@restriction='package']">
    <xsl:with-param name="within">
     <xsl:value-of select="$temp_within"/>
    </xsl:with-param>
   </xsl:apply-templates>
  </xsl:element>
 </xsl:template>

 <!-- template for models -->
 <xsl:template match="definition[@restriction='model']">
  <xsl:param name="within" />
  <!-- create correct full path -->
  <xsl:variable name="temp_within">
   <xsl:choose>
    <xsl:when test="string-length($within) > 0">
     <xsl:value-of select="concat($within,'.',@ident)"/>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="@ident"/>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:variable>
  <xsl:variable name="ID">
   <xsl:value-of select="replace($temp_within,'\.','_')"/>
  </xsl:variable>
  <!-- create model node -->
  <xsl:element name="{@restriction}">
   <xsl:element name="modelID">
```

```
80      <xsl:value-of select="$ID"/>
81    </xsl:element>
82    <xsl:element name="fullPath">
83      <xsl:value-of select="$temp_within"/>
84    </xsl:element>
85    <xsl:element name="modelType">
86      <xsl:value-of select="annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification[./component_reference/@ident='modeltype']/modification_equals/
          component_reference/@ident" />
87    </xsl:element>
88    <xsl:element name="modelName">
89      <xsl:value-of select="annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification[./component_reference/@ident='modelname']/modification_equals/
          string_literal/@value"/>
90    </xsl:element>
91    <xsl:for-each select="extends">
92      <xsl:element name="extends">
93        <xsl:value-of select="@type" />
94      </xsl:element>
95    </xsl:for-each>
96    <!-- solver settings -->
97    <xsl:element name="solver">
98      <xsl:value-of select="annotation/element_modification[./component_reference/@ident='experiment']/
          modification_arguments/element_modification[./component_reference/@ident='__Dymola_Algorithm']/modification_equals/
          string_literal/@value"/>
99    </xsl:element>
100   <!-- stepsize settings -->
101   <xsl:if test="annotation/element_modification[./component_reference/@ident='experiment']/modification_arguments/
        element_modification/component_reference/@ident='__Dymola_fixedstepsize'">
102     <xsl:element name="stepsize">
103       <xsl:value-of select="annotation/element_modification[./component_reference/@ident='experiment']/
          modification_arguments/element_modification[./component_reference/@ident='__Dymola_fixedstepsize']/
          modification_equals/real_literal/@value"/>
104     </xsl:element>
105   </xsl:if>
106   <!-- modeling depth -->
107   <xsl:if test="annotation/element_modification[./component_reference/@ident='ENTIME']/modification_arguments/
        element_modification/component_reference/@ident='modelingDepth'">
108     <xsl:element name="modelingDepth">
109       <xsl:value-of select="concat(annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification[./component_reference/@ident='modelingDepth']/modification_equals/
          integer_literal/@value,annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification[./component_reference/@ident='modelingDepth']/modification_equals/
          real_literal/@value)"/>
110     </xsl:element>
111   </xsl:if>
112   <!-- get interface model -->
113   <xsl:for-each select="annotation/element_modification[./component_reference/@ident='ENTIME']/
        modification_arguments/element_modification[./component_reference/@ident='inheritsInterfacesFrom']">
114     <xsl:element name="inheritsInterfacesFrom">
115       <xsl:variable name="IFM">
116         <xsl:for-each select="./modification_equals//component_reference">
117           <xsl:value-of select="concat(@ident,'_')" />
118         </xsl:for-each>
119       </xsl:variable>
120       <xsl:value-of select="substring($IFM,1,string-length($IFM)-1)"/>
121     </xsl:element>
122   </xsl:for-each>
123   <!-- choose components -->
124   <xsl:apply-templates select="component">
125     <xsl:with-param name="within">
126       <xsl:value-of select="$temp_within"/>
127     </xsl:with-param>
128   </xsl:apply-templates>
129   <!-- get documentation of the model-->
130   <xsl:for-each select="annotation[.//component_reference/@ident='Documentation']">
131     <xsl:element name="annotation">
132       <xsl:value-of select=".//string_literal/@value"/>
133     </xsl:element>
134   </xsl:for-each>
135   <!-- get connections of the model -->
136   <xsl:for-each select="equation/equ_connect">
137     <xsl:element name="connection">
138       <xsl:for-each select="./component_reference">
139         <xsl:element name="connect">
140           <!--<xsl:value-of select="concat(@ident,'.',component_reference/@ident)"/>-->
141           <xsl:choose>
142           <xsl:when test="string-length(component_reference/@ident) > 0">
143             <xsl:value-of select="concat(@ident,'.',component_reference/@ident)"/>
144           </xsl:when>
145           <xsl:otherwise>
146             <xsl:value-of select="@ident"/>
147           </xsl:otherwise>
148         </xsl:choose>
```

```
149       </xsl:element>
150      </xsl:for-each>
151     </xsl:element>
152    </xsl:for-each>
153    <xsl:element name="modelLanguage">Modelica</xsl:element>
154   </xsl:element>
155
156   <!-- create ENTIME specific tag of modeltype e.g. SolutionPatternModel if present -->
157   <!-- for this model -->
158   <xsl:if test="annotation/element_modification[./component_reference/@ident='ENTIME']/modification_arguments/
          element_modification/component_reference/@ident='modeltype'">
159    <xsl:element name="{annotation/element_modification[./component_reference/@ident='ENTIME']/modification_arguments/
          element_modification[./component_reference/@ident='modeltype']/modification_equals/component_reference/@ident}">
160     <xsl:value-of select="$ID"/>
161    </xsl:element>
162   </xsl:if>
163   <!-- for components also -->
164   <xsl:for-each select="./component[./comment/annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification/component_reference/@ident='modeltype']">
165    <xsl:element name="{./comment/annotation/element_modification/modification_arguments/element_modification/
          modification_equals/component_reference/@ident}">
166     <xsl:value-of select="replace(@type,'\.','_')"/>
167    </xsl:element>
168   </xsl:for-each>
169  </xsl:template>
170
171  <!-- template for components -->
172  <xsl:template match="component">
173   <xsl:param name="within" />
174   <!-- Parameters which have ENTIME Annotation -->
175   <xsl:for-each select=".[.//annotation//component_reference/@ident='ENTIME']">
176     <xsl:element name="component">
177      <!-- Parameter -->
178      <xsl:if test="@variability = 'parameter'">
179       <xsl:element name="parameter">
180        <xsl:element name="parameterName">
181         <xsl:value-of select="@ident" />
182        </xsl:element>
183        <!-- create unique parameterID -->
184        <xsl:element name="parameterID">
185         <xsl:value-of select="concat(replace($within,'\.','_'),'_PAR_',@ident)" />
186        </xsl:element>
187        <!-- get Modelica.SIunits and undo common shortening by SI=import... -->
188        <xsl:element name="modelicaType">
189         <xsl:choose>
190          <xsl:when test="starts-with(@type,'SI.')">
191           <xsl:value-of select="replace(@type,'SI.','Modelica.SIunits.')"/>
192             </xsl:when>
193          <xsl:otherwise>
194           <xsl:value-of select="@type"/>
195          </xsl:otherwise>
196         </xsl:choose>
197        </xsl:element>
198        <xsl:element name="parameterDescription">
199          <xsl:value-of select=".//string_comment/string_literal/@value"/>
200        </xsl:element>
201        <xsl:element name="defaultValue">
202          <!-- single value -->
203          <xsl:value-of select="modification_equals/*/@value"/>
204          <!-- vector -->
205          <xsl:for-each select="modification_equals/array/expression_list/*">
206           <xsl:value-of select="@value"/>
207           <xsl:if test="local-name()='comma'">,</xsl:if>
208          </xsl:for-each>
209        </xsl:element>
210        <xsl:for-each select=".//annotation/element_modification[./component_reference/@ident='ENTIME']/
       modification_arguments/element_modification[./modification_equals/bool_literal/@value='true']">
211          <xsl:element name="ENTIME">
212           <xsl:value-of select="./component_reference/@ident" />
213          </xsl:element>
214        </xsl:for-each>
215       </xsl:element>
216      </xsl:if>
217      <!-- System Element models -->
218      <xsl:if test="./comment/annotation/element_modification[./component_reference/@ident='ENTIME']/
       modification_arguments/element_modification[./component_reference/@ident='modeltype']/modification_equals/
       component_reference/@ident = 'SystemElementModel'">
219       <xsl:element name="SystemElementModel">
220        <xsl:element name="SystemElementName">
221         <xsl:value-of select=".//annotation/element_modification[./component_reference/@ident='ENTIME']/
       modification_arguments/element_modification[./component_reference/@ident='SystemElementName']/modification_equals/
       string_literal/@value"/>
222        </xsl:element>
223        <xsl:element name="SystemElementID">
```

```
224          <xsl:value-of select="@ident"/>
225         </xsl:element>
226         <xsl:variable name="baseID">
227          <xsl:value-of select="replace(@type,'\.','_')"/>
228         </xsl:variable>
229         <xsl:element name="baseModelID">
230          <xsl:value-of select="$baseID"/>
231         </xsl:element>
232         <xsl:element name="baseModelPath">
233          <xsl:value-of select="@type"/>
234         </xsl:element>
235         <xsl:for-each select="modification_arguments/element_modification">
236          <xsl:element name="modifier">
237           <xsl:element name="parameterName">
238            <xsl:value-of select="component_reference/@ident"/>
239           </xsl:element>
240           <xsl:element name="parameterID">
241            <xsl:value-of select="concat($baseID,'_PAR_',component_reference/@ident)" />
242           </xsl:element>
243           <xsl:element name="parameterValue">
244            <!-- single value -->
245            <xsl:value-of select="modification_equals/*/@value"/>
246            <!-- vector -->
247            <xsl:for-each select="modification_equals/array/expression_list/*">
248             <xsl:value-of select="@value"/>
249             <xsl:if test="local-name()='comma'">,</xsl:if>
250            </xsl:for-each>
251           </xsl:element>
252          </xsl:element>
253         </xsl:for-each>
254        </xsl:element>
255       </xsl:if>
256       <!-- Environment Element models -->
257       <xsl:if test="./comment/annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification[./component_reference/@ident='modeltype']/modification_equals/
          component_reference/@ident = 'EnvironmentElementModel'">
258        <xsl:element name="EnvironmentElementModel">
259         <xsl:element name="EnvironmentElementName">
260          <xsl:value-of select=".//annotation/element_modification[./component_reference/@ident='ENTIME']/
          modification_arguments/element_modification[./component_reference/@ident='EnvironmentElementName']/
          modification_equals/string_literal/@value"/>
261         </xsl:element>
262         <xsl:element name="EnvironmentElementID">
263          <xsl:value-of select="@ident"/>
264         </xsl:element>
265         <xsl:variable name="baseID">
266          <xsl:value-of select="replace(@type,'\.','_')"/>
267         </xsl:variable>
268         <xsl:element name="baseModelID">
269          <xsl:value-of select="$baseID"/>
270         </xsl:element>
271         <xsl:element name="baseModelPath">
272          <xsl:value-of select="@type"/>
273         </xsl:element>
274         <xsl:for-each select="modification_arguments/element_modification">
275          <xsl:element name="modifier">
276           <xsl:element name="parameterName">
277            <xsl:value-of select="component_reference/@ident"/>
278           </xsl:element>
279           <xsl:element name="parameterID">
280            <xsl:value-of select="concat($baseID,'_PAR_',component_reference/@ident)" />
281           </xsl:element>
282           <xsl:element name="parameterValue">
283            <!-- single value -->
284            <xsl:value-of select="modification_equals/*/@value"/>
285            <!-- vector -->
286            <xsl:for-each select="modification_equals/array/expression_list/*">
287             <xsl:value-of select="@value"/>
288             <xsl:if test="local-name()='comma'">,</xsl:if>
289            </xsl:for-each>
290           </xsl:element>
291          </xsl:element>
292         </xsl:for-each>
293        </xsl:element>
294       </xsl:if>
295      </xsl:element>
296     </xsl:for-each>
297     <!-- Interfaces -->
298     <!-- <xsl:for-each select=".[@type='Modelica.Blocks.Interfaces.RealInput']"> -->
299     <xsl:for-each select=".[contains(@type,'Interfaces')]">
300      <xsl:element name="component">
301       <xsl:element name="interface">
302        <xsl:element name="interfaceName">
303         <xsl:value-of select="@ident" />
```

```
304       </xsl:element>
305       <xsl:element name="interfaceID">
306        <xsl:value-of select="concat(replace($within,'\.','_'),'_IF_',@ident)" />
307       </xsl:element>
308       <xsl:element name="modelicaType">
309        <xsl:value-of select="@type"/>
310       </xsl:element>
311       <xsl:choose>
312        <xsl:when test="contains(@type,'Input')">
313         <xsl:element name="interfaceType">Input</xsl:element>
314        </xsl:when>
315        <xsl:when test="contains(@type,'Output')">
316         <xsl:element name="interfaceType">Output</xsl:element>
317        </xsl:when>
318        <xsl:otherwise>
319         <xsl:element name="interfaceType">Port</xsl:element>
320        </xsl:otherwise>
321       </xsl:choose>
322      </xsl:element>
323     </xsl:element>
324    </xsl:for-each>
325   </xsl:template>
326 </xsl:stylesheet>
```

A.4 Java ModelXML Parser

```
1  package modelXML;
2
3  import java.io.File;
4
5  import javax.xml.parsers.DocumentBuilder;
6  import javax.xml.parsers.DocumentBuilderFactory;
7
8  import org.w3c.dom.Document;
9  import org.w3c.dom.Element;
10 import org.w3c.dom.Node;
11 import org.w3c.dom.NodeList;
12 import org.w3c.dom.Text;
13
14 public class ModelXMLParser {
15
16    private Document document;
17    private int numberOfModels;
18    private NodeList modelNodesList;
19
20    public ModelXMLParser(String xmlFile) throws Exception
21    {
22       DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
23       DocumentBuilder builder = factory.newDocumentBuilder();
24       //Build Document
25       this.document = builder.parse(new File(xmlFile));
26       //Normalize the XML Structure;
27       document.getDocumentElement().normalize();
28       this.numberOfModels = this.getNumberOfModels();
29       this.modelNodesList = this.document.getElementsByTagName("model");
30    }
31
32    public static void main (String args[])
33    {
34    }
35
36    public boolean testModelXML(){
37       Element root = this.document.getDocumentElement();
38       return (root.getNodeName() == "modelxml");
39    }
40
41    public int getNumberOfModels(){
42       return this.document.getElementsByTagName("model").getLength();
43    }
44
45    public String[] getModelIDs () {
46       String[] out = new String[this.numberOfModels];
47       for (int i = 0; i < numberOfModels; i++){
48           Node node = this.modelNodesList.item(i);
49           Element element = (Element) node;
50           if (node.getNodeType() == Node.ELEMENT_NODE) {
51              out[i]=element.getElementsByTagName("modelID").item(0).getTextContent();
52            }
53       }
54       return out;
55    }
```

```
56
57    private Node getModelNode(String modelID){
58       Node node;
59       for (int i=0; i < this.numberOfModels; i++){
60          node = this.getNode("modelID",this.modelNodesList.item(i).getChildNodes());
61          if (getNodeValue(node).equalsIgnoreCase(modelID)){
62             return this.modelNodesList.item(i);
63          }
64       }
65    return null;
66    }
67
68    private NodeList getComponentNodes (String modelID, String componentTagName){
69       Node modelNode = getModelNode(modelID);
70       return ((Element)modelNode).getElementsByTagName(componentTagName);
71    }
72
73    public String getComponentInfo (String modelID, String componentTag, String componentID, String tagName,
          String componentIDTag){
74       NodeList componentNodes = this.getComponentNodes(modelID,componentTag);
75       int number=componentNodes.getLength();
76       Node node;
77       for (int i=0; i < number; i++){
78          node = this.getNode(componentIDTag,componentNodes.item(i).getChildNodes());
79          if (this.getNodeValue(node).equalsIgnoreCase(componentID)){
80             return this.getNodeValue(tagName,componentNodes.item(i).getChildNodes());
81          }
82       }
83       return "";
84    }
85    public String getComponentInfo (String modelID, String componentTag, String componentID, String tagName){

86       return getComponentInfo (modelID, componentTag, componentID, tagName, componentTag+"ID");
87    }
88
89    public String[] getComponentIDs (String modelID,String componentTag, String componentIDTag){
90       NodeList componentNodes = this.getComponentNodes(modelID,componentTag);
91       int number=componentNodes.getLength();
92       String[] out = new String[number];
93       for (int i = 0; i < number; i++){
94           Node node = componentNodes.item(i);
95           Element element = (Element) node;
96           if (node.getNodeType() == Node.ELEMENT_NODE) {
97              out[i]=element.getElementsByTagName(componentIDTag).item(0).getTextContent();
98            }
99       }
100      return out;
101   }
102
103   public String[] getComponentIDs (String modelID,String componentTag){
104      return getComponentIDs (modelID, componentTag, componentTag+"ID");
105   }
106
107   public String getComponentInterfaceInfo (String modelID, String interfaceID, String tagName){
108      return getComponentInfo(modelID,"interface",interfaceID,tagName);
109   }
110
111   public String[] getComponentParameterIDs (String modelID){
112      return getComponentIDs (modelID,"parameter","parameterID");
113   }
114
115   public String getComponentParameterInfo (String modelID, String parameterID, String tagName){
116      return getComponentInfo(modelID,"parameter",parameterID,tagName);
117   }
118
119   public String[] getComponentSystemElementModelIDs (String modelID){
120      return getComponentIDs (modelID,"SystemElementModel","SystemElementID");
121   }
122
123   public String getComponentSystemElementModelInfo (String modelID, String systemElementID, String tagName)
      {
124      return getComponentInfo (modelID, "SystemElementModel", systemElementID, tagName, "SystemElementID");
125   }
126
127   public String[][] getComponentSystemElementModelModifier (String modelID, String systemElementID){
128      NodeList componentSystemElementModelNodes = this.getComponentNodes(modelID,"SystemElementModel");
129      int number=componentSystemElementModelNodes.getLength();
130      Node node;
131
132      for (int i=0; i < number; i++){
133         node = this.getNode("SystemElementID",componentSystemElementModelNodes.item(i).getChildNodes());
134         if (this.getNodeValue(node).equalsIgnoreCase(systemElementID)){
135            NodeList modifierNodes=((Element)componentSystemElementModelNodes.item(i)).getElementsByTagName
      ("modifier");
```

```
136             String[][] out=new String[modifierNodes.getLength()][2];
137
138             for (int z=0; z < modifierNodes.getLength(); z++){
139                out[z][0]=this.getNodeValue("parameterName",modifierNodes.item(z).getChildNodes());
140                out[z][1]=this.getNodeValue("parameterValue",modifierNodes.item(z).getChildNodes());
141             }
142             return out;
143          }
144       }
145       String[][] nullString={{""},{""}};
146       return nullString;
147    }
148
149    public String[][] getModelConnections (String modelID){
150       Node modelNode=getModelNode(modelID);
151       NodeList connectionNodes=((Element)modelNode).getElementsByTagName("connection");
152       String[][] out=new String[connectionNodes.getLength()][2];
153
154       for (int z=0; z < connectionNodes.getLength(); z++){
155          out[z][0]=this.getNodeValue(connectionNodes.item(z).getChildNodes().item(1));
156          out[z][1]=this.getNodeValue(connectionNodes.item(z).getChildNodes().item(3));
157       }
158       return out;
159    }
160
161    public String getModelInfo (String modelID, String tagName){
162       return this.getNodeValue(tagName,this.getModelNode(modelID).getChildNodes());
163    }
164
165    public String getSolver (String modelID){
166       return this.getModelInfo (modelID,"solver");
167    }
168
169    public String getStepSize (String modelID){
170       return this.getModelInfo (modelID,"stepsize");
171    }
172
173    public String getFullPath (String modelID){
174       return this.getModelInfo (modelID,"fullPath");
175    }
176
177    public String getModelType (String modelID){
178       return this.getModelInfo (modelID,"modelType");
179    }
180
181    public String getModelName (String modelID){
182       return this.getModelInfo (modelID,"modelName");
183    }
184
185    public String getModelingDepth (String modelID){
186       return this.getModelInfo (modelID,"modelingDepth");
187    }
188
189    ////===Helper methods from:===////
190    ////===http://www.drdobbs.com/jvm/easy-dom-parsing-in-java/231002580===////
191    protected Node getNode(String tagName, NodeList nodes) {
192        for ( int x = 0; x < nodes.getLength(); x++ ) {
193            Node node = nodes.item(x);
194            if (node.getNodeName().equalsIgnoreCase(tagName)) {
195                return node;
196            }
197        }
198        return null;
199    }
200
201    protected String getNodeValue( Node node ) {
202        NodeList childNodes = node.getChildNodes();
203        for (int x = 0; x < childNodes.getLength(); x++ ) {
204            Node data = childNodes.item(x);
205            if ( data.getNodeType() == Node.TEXT_NODE )
206                return data.getNodeValue();
207        }
208        return "";
209    }
210
211    protected String getNodeValue(String tagName, NodeList nodes ) {
212        for ( int x = 0; x < nodes.getLength(); x++ ) {
213            Node node = nodes.item(x);
214            if (node.getNodeName().equalsIgnoreCase(tagName)) {
215                NodeList childNodes = node.getChildNodes();
216                for (int y = 0; y < childNodes.getLength(); y++ ) {
217                    Node data = childNodes.item(y);
218                    if ( data.getNodeType() == Node.TEXT_NODE )
219                        return data.getNodeValue();
```

```
220                 }
221             }
222         }
223         return "";
224     }
225 
226     protected void addNode(String tagName, String value, Node parent) {
227         Document dom = parent.getOwnerDocument();
228 
229         // Create a new Node with the given tag name
230         Node node = dom.createElement(tagName);
231 
232         // Add the node value as a child text node
233         Text nodeVal = dom.createTextNode(value);
234         Node c = node.appendChild(nodeVal);
235 
236         // Add the new node structure to the parent node
237         parent.appendChild(c);
238     }
239 
240     protected void setNodeValue(String tagName, String value, NodeList nodes) {
241         Node node = getNode(tagName, nodes);
242         if ( node == null )
243             return;
244 
245         // Locate the child text node and change its value
246         NodeList childNodes = node.getChildNodes();
247         for (int y = 0; y < childNodes.getLength(); y++ ) {
248             Node data = childNodes.item(y);
249             if ( data.getNodeType() == Node.TEXT_NODE ) {
250                 data.setNodeValue(value);
251                 return;
252             }
253         }
254     }
255 }
```

Das Heinz Nixdorf Institut – Interdisziplinäres Forschungszentrum für Informatik und Technik

Das Heinz Nixdorf Institut ist ein Forschungszentrum der Universität Paderborn. Es entstand 1987 aus der Initiative und mit Förderung von Heinz Nixdorf. Damit wollte er Ingenieurwissenschaften und Informatik zusammenführen, um wesentliche Impulse für neue Produkte und Dienstleistungen zu erzeugen. Dies schließt auch die Wechselwirkungen mit dem gesellschaftlichen Umfeld ein.

Die Forschungsarbeit orientiert sich an dem Programm „Dynamik, Mobilität, Vernetzung: Eine neue Schule des Entwurfs der technischen Systeme von morgen“. In der Lehre engagiert sich das Heinz Nixdorf Institut in Studiengängen der Informatik, der Ingenieurwissenschaften und der Wirtschaftswissenschaften.

Heute wirken am Heinz Nixdorf Institut neun Professoren mit insgesamt 150 Mitarbeiterinnen und Mitarbeitern. Pro Jahr promovieren hier etwa 20 Nachwuchswissenschaftlerinnen und Nachwuchswissenschaftler.

Heinz Nixdorf Institute – Interdisciplinary Research Centre for Computer Science and Technology

The Heinz Nixdorf Institute is a research centre within the University of Paderborn. It was founded in 1987 initiated and supported by Heinz Nixdorf. By doing so he wanted to create a symbiosis of computer science and engineering in order to provide critical impetus for new products and services. This includes interactions with the social environment.

Our research is aligned with the program “Dynamics, Mobility, Integration: Enroute to the technical systems of tomorrow“. In training and education the Heinz Nixdorf Institute is involved in many programs of study at the University of Paderborn. The superior goal in education and training is to communicate competencies that are critical in tomorrows economy.

Today nine Professors and 150 researchers work at the Heinz Nixdorf Institute. Per year approximately 20 young researchers receive a doctorate.

Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 351 BRENNER, C.: Szenariobasierte Synthese verteilter mechatronischer Systeme. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 351, Paderborn, 2016 – ISBN 978-3-942647-70-0

Bd. 352 WALL, M.: Systematik zur technologieinduzierten Produkt- und Technologieplanung. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 352, Paderborn, 2016 – ISBN 978-3-942647-71-7

Bd. 353 CORD-LANDWEHR, A.: Selfish Network Creation - On Variants of Network Creation Games. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 353, Paderborn, 2016 – ISBN 978-3-942647-72-4

Bd. 354 ANACKER, H.: Instrumentarium für einen lösungsmusterbasierten Entwurf fortgeschrittener mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 354, Paderborn, 2016 – ISBN 978-3-942647-73-1

Bd. 355 RUDTSCH, V.: Methodik zur Bewertung von Produktionssystemen in der frühen Entwicklungsphase. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 355, Paderborn, 2016 – ISBN 978-3-942647-74-8

Bd. 356 SÖLLNER, C.: Methode zur Planung eines zukunftsfähigen Produktportfolios. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 356, Paderborn, 2016 – ISBN 978-3-942647-75-5

Bd. 357 AMSHOFF, B.: Systematik zur musterbasierten Entwicklung technologieinduzierter Geschäftsmodelle. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 357, Paderborn, 2016 – ISBN 978-3-942647-76-2

Bd. 358 LÖFFLER, A.: Entwicklung einer modellbasierten In-the-Loop-Testumgebung für Waschautomaten. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 358, Paderborn, 2016 – ISBN 978-3-942647-77-9

Bd. 359 LEHNER, A.: Systematik zur lösungsmusterbasierten Entwicklung von Frugal Innovations. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 359, Paderborn, 2016 – ISBN 978-3-942647-78-6

Bd. 360 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 12. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 8. und 9. Dezember 2016, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 360, Paderborn, 2016 – ISBN 978-3-942647-79-3

Bd. 361 PETER, S.: Systematik zur Antizipation von Stakeholder-Reaktionen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 361, Paderborn, 2016 – ISBN 978-3-942647-80-9

Bd. 362 ECHTERHOFF, O.: Systematik zur Erarbeitung modellbasierter Entwicklungsaufträge. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 362, Paderborn, 2016 – ISBN 978-3-942647-81-6

Bd. 363 TSCHIRNER, C.: Rahmenwerk zur Integration des modellbasierten Systems Engineering in die Produktentstehung mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 363, Paderborn, 2016 – ISBN 978-3-942647-82-3

Bd. 364 KNOOP, S.: Flachheitsbasierte Positionsregelungen für Parallelkinematiken am Beispiel eines hochdynamischen hydraulischen Hexapoden. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 364, Paderborn, 2016 – ISBN 978-3-942647-83-0

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 365 KLIEWE, D.: Entwurfssystematik für den präventiven Schutz Intelligenter Technischer Systeme vor Produktpiraterie. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 365, Paderborn, 2017 – ISBN 978-3- 942647-84-7

Bd. 366 IWANEK, P.: Systematik zur Steigerung der Intelligenz mechatronischer Systeme im Maschinen- und Anlagenbau. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 366, Paderborn, 2017 – ISBN 978-3-942647-85-4

Bd. 367 SCHWEERS, C.: Adaptive Sigma-Punkte-Filter-Auslegung zur Zustands- und Parameterschätzung an Black-Box-Modellen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 367, Paderborn, 2017 – ISBN 978-3-942647-86-1

Bd. 368 SCHIERBAUM, T.: Systematik zur Kostenbewertung im Systementwurf mechatronischer Systeme in der Technologie Molded Interconnect Devices (MID). Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 368, Paderborn, 2017 – ISBN 978-3-942647-87-8

Bd. 369 BODDEN, E.; DRESSLER, F.; DUMITRESCU, R.; GAUSEMEIER, J.; MEYER AUF DER HEIDE, F.; SCHEYTT, C.; TRÄCHTLER, A. (Hrsg.): Intelligente technische Systeme. Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 369, Paderborn, 2017 – ISBN 978-3-942647-88-58

Bd. 370 KÜHN, A.: Systematik zur Release-Planung intelligenter technischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 370, Paderborn, 2017 – ISBN 978-3-942647-89-2

Bd. 371 REINOLD, P.: Integrierte, selbstoptimierende Fahrdynamikregelung mit Einzelradaktorik. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 371, Paderborn, 2017 – ISBN 978-3-942647-90-8

Bd. 372 BÄUMER, F. S.: Indikatorbasierte Erkennung und Kompensation von ungenauen und unvollständig beschriebenen Softwareanforderungen. Dissertation, Fakultät für Wirtschaftswissenschaften, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 372, Paderborn, 2017 – ISBN 978-3-942647-91-5

Bd. 373 ECKELT, D.: Systematik zum innovationsorientierten Intellectual Property Management. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 373, Paderborn, 2017 – ISBN 978-3-942647-92-2

Bd. 374 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 13. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 23. und 24. November 2017, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 374, Paderborn, 2017 – ISBN 978-3-942647-93-9

Bd. 375 WESTERMANN, T.: Systematik zur Reifegradmodell-basierten Planung von Cyber-Physical Systems des Maschinenund Anlagenbaus. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 375, Paderborn, 2017 – ISBN 978-3-942647-94-6

Bd. 376 JÜRGENHAKE, C.: Systematik für eine prototypenbasierte Entwicklung mechatronischer Systeme in der Technologie MID (Molded Interconnect Devices). Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 376, Paderborn, 2017 – ISBN 978-3-942647-95-3

Bd. 377 WEBER, J.: Modellbasierte Werkstück und Werkzeugpositionierung zur Reduzierung der Zykluszeit in NC-Programmen. Dissertation, Fakultät für Wirtschaftswissenschaften, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 377, Paderborn, 2018 – ISBN 978-3-942647-96-0

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn