Wiederverwendbare Simulationsmodelle für die domänen- und disziplinübergreifende Produktentwicklung

Von der Fakultät Konstruktions-, Produktions- und Fahrzeugtechnik
der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

Vorgelegt von

Dipl.-Ing. Verena Voß

aus Düren-Birkesdorf

Hauptberichter: Prof. Dr.-Ing. Dr. h.c. mult. A. Verl
Mitberichter: Prof. Dr.-Ing. C. Brecher

Tag der mündlichen Prüfung: 31.10.2012

Institut für Steuerungstechnik der Werkzeugmaschinen
und Fertigungseinrichtungen
der Universität Stuttgart

2012

ISW/IPA Forschung und Praxis

Berichte aus dem Institut für Steuerungstechnik
der Werkzeugmaschinen und Fertigungseinrichtungen
der Universität Stuttgart und dem Fraunhofer-Institut für
Produktionstechnik und Automatisierung IPA

Herausgeber: Prof. Dr.-Ing. Dr. h.c. mult. A. Verl

Institut für Steuerungstechnik
der Werkzeugmaschinen und Fertigungseinrichtungen

Fraunhofer IPA

Verena Voß

Wiederverwendbare Simulationsmodelle für die domänen- und disziplinübergreifende Produktentwicklung

Nr. 192

JOST-JETTER VERLAG
Fachverlag · 71296 Heimsheim

D 93

ISBN 978-3-939890-98-0
Jost Jetter Verlag, Heimsheim

Druck: printsystem GmbH, Heimsheim

Geleitwort des Herausgebers

In der Reihe „ISW/IPA Forschung und Praxis“ wird fortlaufend über Forschungsergebnisse des Instituts für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart (ISW) und des Fraunhofer Instituts für Produktionstechnik und Automatisierung (IPA) berichtet. Die Institute stehen unter gemeinsamer Leitung von Professor Alexander Verl und beschäftigen sich sowohl im Bereich der Grundlagenforschung als auch im Bereich der angewandten Forschung mit der Weiterentwicklung und Optimierung der Automatisierung in der Produktionstechnik. Fraunhofer IPA und ISW repräsentieren Deutschlands größte Forschungseinrichtung im Bereich der Produktionsautomatisierung.

Besonderes Augenmerk wird auf die Systeme Werkzeugmaschine und Roboter gelegt. Die Arbeiten des ISW konzentrieren sich im Besonderen auf die Bereiche Numerische Steuerungstechnik, Bewegungssteuerung, Planungs- und Leitsysteme, Softwaretechnik, Simulationstechnik und Antriebstechnik. Am Fraunhofer IPA konzentriert man sich im Bereich der Automatisierungstechnik auf die Themen: Servicerobotik, Industrierobotik, Montageautomatisierung, Orthopädiesysteme, Prozessautomatisierung (insbesondere im Lifesciencebereich), Messtechnik, Prüftechnik, Bildverarbeitung sowie die Automatisierung in der Reinst- und Mikroproduktion. Dabei stehen Grundlagenforschung und anwenderorientierte Entwicklung in einem stetigen Austausch, wodurch ein ständiger Technologietransfer zur Praxis sichergestellt wird.

Die Buchreihe erscheint in zwangloser Folge und stützt sich auf Berichte über abgeschlossene Forschungsarbeiten und Dissertationen. Sie soll dem Ingenieur bei der Weiterbildung dienen und ihm Hilfestellungen zur Lösung spezifischer Probleme geben. Für den Studierenden bietet sie eine Möglichkeit zur Wissensvertiefung. Sie bleibt damit unter erweitertem Namen und in der inzwischen dritten Generation in der bewährten Konzeption, die ihr die Vorgänger am ISW, Professor Gottfried Stute (1972 - 1982) und Professor Günter Pritschow (1984 - 2005), gegeben haben.

Der vorliegende Band beschäftigt sich mit einer neuartigen Methode zur disziplin- und domänenübergreifenden Wiederverwendung von Simulationsmodellen mit dem Ziel, einen wirtschaftlichen Einsatz der Simulation zu ermöglichen sowie die Effizienz in der Entwicklung insgesamt zu erhöhen. Dazu wird ein gemeinsames Meta-Metamodell für unterschiedliche Domänen und Disziplinen entwickelt. Weiterhin werden die zur Nutzung und Integration dieses Meta-Metamodells benötigten Methoden sowie ein Werkzeug erarbeitet. Diese werden prototypisch realisiert und an einem konkreten Anwendungsbeispiel aus dem Bereich des Werkzeugmaschinenbaus validiert.

Der Herausgeber dankt der Druckerei für die drucktechnische Betreuung und dem Jost Jetter Verlag für die Aufnahme der Reihe in sein Lieferprogramm.

A. Verl

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin am Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW) der Universität Stuttgart.

Herrn Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl danke ich herzlich für seine Unterstützung und die wertvollen Anregungen, die zum Gelingen dieser Arbeit wesentlich beigetragen haben, sowie für die Übernahme des Hauptberichtes. Herrn Prof. Dr.-Ing. Christian Brecher danke ich herzlich für seine Bereitschaft, den Mitbericht zu übernehmen. Für die Übernahme des Vorsitzes der Promotionskommission danke ich darüber hinaus Herrn Prof. Dr.-Ing. Dr. h.c. Karl-Heinz Wehking.

Ich danke dem Bundesministerium für Bildung und Forschung für die Förderung des Projektes „Wieder verwendbare Modelle für die virtuelle wissensbasierte Produktentwicklung (WieMod)“, aus dem diese Arbeit hervorgegangen ist. Den Projektpartnern *Astro- und Feinwerktechnik Adlershof GmbH*, dem *Deutschen Zentrum für Luft- und Raumfahrt*, der *itemis AG* sowie der *Industriellen Steuerungstechnik GmbH* danke ich für die gute Zusammenarbeit. Herrn Dipl.-Ing. Niko Croon danke ich dafür, dass er mir sein Modell des Kugelgewindetriebs zur Validierung der im Rahmen meiner Arbeit entwickelten Methoden zur Verfügung gestellt hat.

Weiterhin danke ich allen aktuellen und ehemaligen Mitarbeitern des Institutes für die zahlreichen Diskussionen und das angenehme Arbeitsklima, insbesondere den Kollegen aus der ehemaligen Abteilung 1. Den Herren Dr.-Ing. Vladimir Oglodin, Dr.-Ing. Armin Lechler und Dr.-Ing. Dipl.-Inform. Markus Voß danke ich des Weiteren sehr für die kritische Durchsicht meiner Arbeit.

Ein besonderer Dank gilt meinen Eltern Gertrud und Wolfgang Müller für die Ermöglichung meiner Ausbildung und ihre unermüdliche moralische Unterstützung. Danke für den Rückhalt, den Ihr mir mein Leben lang gegeben habt und dafür, dass Ihr immer an mich geglaubt habt. Meinem Bruder Ulf Müller und seiner Freundin Kathrin Keul danke ich für die vielen aufmunternden Gespräche, vor allen Dingen während der diversen Sinnkrisen, die die Erstellung dieser Arbeit mit sich gebracht hat.

Nicht zuletzt danke ich meinem geliebten Ehemann Markus Voß für die vielen fachlichen Diskussionen, die liebevolle Unterstützung, den seelischen Beistand und das Verständnis für die vielen geopferten Wochenenden.

Diese Arbeit widme ich meinem Großvater Matthias Schmitz, der sehr stolz darauf war, dass seine Enkelin als Ingenieurin in seine Fußstapfen getreten ist, aber leider den Abschluss meiner Promotion nicht mehr miterleben durfte.

Verena Voß

Inhaltsverzeichnis

Abkürzungen

ANSYS	ANalysis SYStem
API	Application Programming Interface
AQUIMO	Adaptierbares Modellierungswerkzeug und Qualifizierungsprogramm für den Aufbau firmenspezifischer mechatronischer Engineeringprozesse
ASSERT	Automated proof-based System and Software Engineering for Real-Time applications
AutomationML	Automation Markup Language
AUTOSAR	AUTomotive Open System ARchitecture
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAEX	Computer Aided Engineering Exchange
CAP	Computer Aided Planning
COLLADA	COLLAborative Design Activity
CORBA	Common Object Request Broker Architecture
DOM	Document Object Model
DOM4J	Document Object Model for Java
ECU	Electronic Control Unit
EDV	Elektronische Datenverarbeitung
E/E	Elektronik/Elektrik
EMF	Eclipse Modeling Framework
HIS	Hersteller Initiative Software
HLA	High Level Architecture
HTML	Hyper Text Markup Language
IDE	Integrated Design Environment
IDL	Interface Definition Language
INCOSE	International Council On Systems Engineering
JAXP	Java API for XML Processing
JVM	Java Virtual Machine
LIFO	Last In First Out
MDT	Model Development Tool
MMI	Matlab Model Interface
MOF	Meta Object Facility
MWE	Modeling Workflow Engine

oAW	Open Architecture Ware
OMG	Object Management Group
OMT	Object Model Template
RIF	Requirement Interchange Format
RTE	Runtime Environment
RTI	Runtime Infrastructure
SAX	Simple API for XML
SDAI	Standard Data Interface
SDO	Service Data Object
SE	Systems Engineering
SMP2	Simulation Model Portability 2
SOA	Service Oriented Architecture
STEP	Standard for the Exchange of Product Model Data
UML	Unified Modeling Language
UUID	Universal Unique Identifier
XHTML	eXtended HTML
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Formelzeichen

$A(\omega)$	Betrag des Frequenzganges
f	Frequenz einer Sinusschwingung
$G(j\omega)$	Frequenzgang
m	Anzahl der Transformationen
n	Anzahl der Simulationswerkzeuge
T	Schwingungsdauer einer Sinusschwingung
η	Wirkungsgrad einer Vorgehensweise zur Modelltransformation
$\varphi(\omega)$	Phase des Frequenzganges
Φ	Winkel
ω	Frequenz im Bode-Diagramm

Abstract

For the efficient development of customized products the close collaboration of different domains and discipline is indispensable. The product development process in every domain (mechanical engineering, aerospace, automotive engineering etc.) takes place in several disciplines. Successful development projects are based on the bundling of different competences in interdisciplinary teams. Presently the collaboration of different disciplines is hindered by incompatible, uncoordinated procedures, processes or software tools. For the systematic collaboration however, the complete and automatic exchange of digital data is an important requirement.

In the domain of mechanical engineering for example the disciplines mechanics, electrics and software engineering are involved in the development of a new product. In each discipline various simulation tools are utilized in the course of product development, e.g. Matlab/Simulink, Adams, ANSYS, Virtuos etc. in mechanics, for riskless, virtual testing and optimizing in early stages. This way, in the further course of product design, the necessary time for prototypes and testing can be reduced.

Multiple different models can be created in each simulations tool, such as kinematics models, dynamics models, finite-elements models, multi-body systems and material flow models. Each type of simulation model represents a specific characteristic of a product, so its different aspects can be tested virtually.

This way, in the course of product development, many different simulation models can occur in only one single domain. When considering various domains the multitude of models increases. Most models are incompatible due to different data types, which means, that they can not be re-used in varying domains and disciplines.

In the domain of mechanical engineering simulation is used amongst others for optimization of design or for testing. In the development of mechatronic systems simulation is utilized mainly in large, but rarely in small and medium-sized enterprises (SME). The reason for this fact is mainly the high investment cost, which includes purchase, introduction, and training. Often the effort for the creation of a model is only profitable, when it can be re-used several times. Due to the incompatibility of domain- and discipline-specific simulation tools however, this is not always possible. For re-using a model, which has been created in one specific simulations tool in a different simulation tool, it

usually has to be re-modeled manually. This additional time and effort makes simulation inefficient from an economical point of view.

For an earlier amortization of the effort for the creation of simulation models in future the re-usability of simulation models throughout several domains and disciplines and hence for various simulation tools is aspired. Apart from economic efficiency, re-usability of simulation models can also increase the quality of simulation. The gain of quality results from the fact, that this way a simulation model can be tested in varying simulation tools and thereby under different aspects. This repeated testing leads to well-proven simulation models, which are of benefit to every further simulation model. The advantages and strong points of different simulations tools can be used to full capacity when models can be exchanged without additional effort.

To enable the profitable deployment of simulation and to increase efficiency in product development altogether, in this thesis a concept and methods are developed for multi-domain and interdisciplinary re-use of simulation models. Therefore a common meta-metamodel for various domains and disciplines is designed. Furthermore the required methods and a tool for its utilization and integration are developed. These are prototypically realized and validated for a specific example from the field of machine tools.

1 Einleitung

Die effiziente Entwicklung qualitativ hochwertiger, kundenindividueller Produkte macht eine enge Zusammenarbeit verschiedener Domänen und Disziplinen unabdingbar. Die Entwicklungsarbeiten jeder Domäne (Maschinenbau, Luft- und Raumfahrt, Automobilbau etc.) werden in mehreren Disziplinen durchgeführt. Erfolgreiche Produktentwicklungsprojekte basieren auf der Bündelung verschiedener Kompetenzen in interdisziplinären Entwicklungsteams. Die Zusammenarbeit verschiedener Disziplinen wird derzeit durch inkompatible, nicht aufeinander abgestimmte Vorgehensmodelle, Prozesse oder verwendeter Softwarewerkzeuge erschwert. Für eine systematische Zusammenarbeit ist hingegen ein vollständiger, automatischer Austausch digitaler Daten eine wichtige Voraussetzung. /1, 2, 3, 4/

In der Domäne Maschinenbau sind beispielsweise die Disziplinen Mechanik, Elektrik und Softwaretechnik an der Entwicklung eines neuen Produktes beteiligt. In jeder Disziplin werden im Rahmen der Produktentwicklung verschiedene Simulationswerkzeuge (Disziplin Mechanik: Matlab/Simulink, Adams, ANSYS, Virtuos etc.) eingesetzt, mit deren Hilfe die Produkte bereits in frühen Phasen des Entwicklungsprozesses gefahrlos virtuell getestet und optimiert werden können. So kann im weiteren Verlauf der Produktentstehung die benötigte Zeit für Prototypen und Versuche reduziert werden. /5, 6/

In den unterschiedlichen Simulationswerkzeugen kann eine Vielzahl unterschiedlicher Simulationsmodelle (Kinematikmodelle, Dynamikmodelle, Finite-Elemente-Modelle, Mehrkörpermodelle, Materialflussmodelle etc.) erzeugt werden. Jede Art von Simulationsmodell kann verschiedene Eigenschaften eines Produktes abbilden, so dass dessen unterschiedliche Aspekte in der jeweiligen Simulation virtuell erprobt werden. /3, 7/

Somit können im Rahmen der Produktentwicklung bereits in einer einzigen Domäne sehr viele unterschiedliche Simulationsmodelle entstehen. Domänenübergreifend ist die Modellvielfalt dementsprechend noch umfangreicher. Aufgrund der unterschiedlichen Datenformate der Modelle sind diese meist nicht kompatibel zueinander und können zwischen den verschiedenen Domänen und Disziplinen nicht direkt ausgetauscht und somit nicht durchgängig genutzt werden (Bild 1-1). /8, 9/

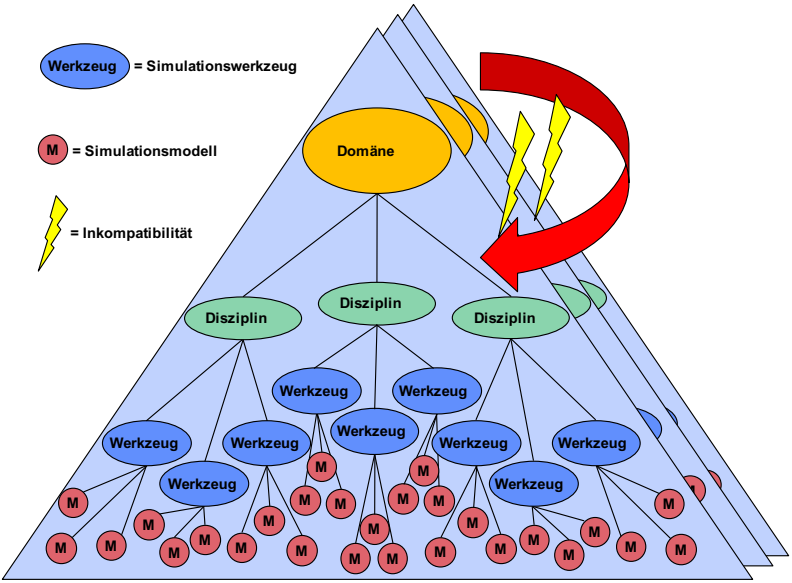


Bild 1-1: Modellvielfalt innerhalb der Domänen /9/

Die Simulation wird in der Domäne Maschinenbau unter anderem zur Optimierung der Konstruktion sowie im Rahmen von Tests, wie z.B. der virtuellen Inbetriebnahme eingesetzt. Im Rahmen der Produktentwicklung wird die Simulation hauptsächlich in größeren, hingegen kaum in mittelständischen Unternehmen eingesetzt. Dies ist vor allen Dingen mit den hohen Investitionskosten (Erwerb, Einführung, Schulung) zu begründen. Der Aufwand für die Modellerstellung rentiert sich oft erst, wenn ein einmal erstelltes Modell mehrfach wiederverwendet werden kann. Auf Grund der noch vorherrschenden Inkompatibilität domänen- und disziplinspezifischer Simulationswerkzeuge, kann ein Simulationsmodell, das in einem spezifischen Simulationswerkzeug erstellt wurde, in einem anderen Simulationswerkzeug jedoch nicht direkt wiederverwendet werden. Die so entstehenden Systembrüche können nur mit Unterstützung der Systemhersteller und aufwändiger Datenaufbereitung überwunden werden. Um ein Simulationsmodell in einem anderen Simulationswerkzeug dennoch nutzen zu können, muss dieses in der Regel in dem anderen Simulationswerkzeug manuell nachmodelliert werden. Dieser Zusatzaufwand macht die Simulation unwirtschaftlich. /8, 10, 11/

Um zukünftig den Aufwand zur Erstellung von Simulationsmodellen früher zu amortisieren, wird eine domänen- und disziplinübergreifende Wiederverwendung von Simulationsmodellen für unterschiedliche Simulationswerkzeuge angestrebt. Die Wiederverwendbarkeit von Simulationsmodellen kann neben der Wirtschaftlichkeit auch die Qualität der Simulation erhöhen. Die Qualitätssteigerung ergibt sich unter anderem dadurch, dass ein Simulationsmodell in mehreren Simulationswerkzeugen und darin unter verschiedenen Gesichtspunkten getestet und validiert werden kann. Durch die mehrfache Prüfung entstehen erprobte Simulationsmodelle mit höherer Genauigkeit, die durch erneute Wiederverwendung jedem darauf basierenden Modell zugutekommen. Auch kann die Entwicklungsqualität durch frühzeitige interdisziplinäre Zusammenarbeit maßgeblich erhöht werden. Die Vorzüge und Stärken der verschiedenen Simulationswerkzeuge können so wesentlich effektiver genutzt werden, da die Modelle zwischen den Werkzeugen ohne Zusatzaufwand ausgetauscht werden können. /10, 12/

Um den wirtschaftlichen Einsatz der Simulation zu ermöglichen sowie die Effizienz in der Entwicklung insgesamt zu erhöhen, wird in dieser Arbeit ein Konzept für die domänen- und disziplinübergreifende Wiederverwendung von Simulationsmodellen entwickelt, welches im Folgenden als „WieMod“ bezeichnet wird. Dazu wird ein gemeinsames Meta-Metamodell für die unterschiedlichen Domänen und Disziplinen entwickelt. Weiterhin werden die zur Nutzung und Integration dieses Meta-Metamodells benötigten Methoden sowie ein Werkzeug erarbeitet. Diese werden prototypisch realisiert und an einem konkreten Anwendungsbeispiel aus dem Bereich des Werkzeugmaschinenbaus validiert.

2 Begriffsdefinitionen und Anforderungen

In diesem Kapitel werden die der Arbeit zugrundeliegenden Begriffe definiert und Anforderungen an das zu entwickelnde Konzept erarbeitet.

2.1 Begriffsdefinitionen

2.1.1 Modell und Metamodell

Der Begriff **Modell** ist nach ISO definiert als die formale Beschreibung einer endlichen Menge von Fakten, Konzepten oder Instruktionen in Übereinstimmung mit Regeln, die zuvor explizit vereinbart wurden. /13/

Ein **Simulationsmodell** ist ein spezielles Modell, das für die Zwecke der Simulation implementiert wird und in einem Simulationswerkzeug interpretiert werden kann. Drei Arten der Modellbeschreibung können bei Simulationsmodellen unterschieden werden. Die **graphikorientierte Modellbeschreibung** verwendet standardisierte Symbole oder nutzerdefinierte Graphiken als Bausteine zur Darstellung der Aufbaustruktur des Modells und kann durch Parametereingaben und textuelle Beschreibungen ergänzt werden. Ein typisches Beispiel für eine graphikorientierte Modellbeschreibung sind blockschaltbildbasierte Simulationsmodelle mit graphischer Repräsentation. Bei der **parameterorientierten Modellbeschreibung** werden nur parametrisch veränderbare Modelle vorausgesetzt. Aus diesem Grund ist diese Art der Modellbeschreibung von geringer Bedeutung und in der Praxis kaum anzutreffen. Die **sprachorientierte Modellbeschreibung** benutzt Simulationssprachen, mit deren Hilfe beliebige Systeme und Prozesse nachgebildet werden können. Diese Art der Modellbeschreibung erfolgt rein textuell und ohne graphische Repräsentation. /14/

Der Begriff **Metamodell** ist nach Strahringer wie folgt definiert: „Ein Metamodell ist ein Modell eines Modells, wobei es sich bei dem übergeordneten Modell um ein sprachliches Beschreibungsmodell handelt, das die Sprache, in der das untergeordnete Modell formuliert ist, abbildet.“ /15/

Metamodelle enthalten Wissen über die Eigenschaften eines bestimmten Modells bzw. Modelltyps und bilden somit dessen Grundlage. Bei einem Metamodell handelt es sich immer um eine formale Darstellung der Modelleigenschaften. Der prinzipielle Aufbau

eines bestimmten Modells wird im Metamodell beschrieben. Das Metamodell zeigt die Komponenten (Modellbausteine) eines bestimmten Modells und deren Beziehungen untereinander auf und bildet somit den „Bauplan“ des Modells. Die Modellierungssprache, mit der ein Modell beschrieben wird, wird im Metamodell definiert. /16/

Um den Zusammenhang des Begriffes „Modell” im Rahmen der Simulation zu verdeutlichen, folgt eine Einführung in die Simulation und den Simulationsprozess im Allgemeinen.

2.1.2 Simulation und Simulationsprozess

Die **Simulation** ist eine Vorgehensweise zur Analyse von Systemen, die auf Grund ihrer Komplexität für die theoretische oder formelmäßige Behandlung ungeeignet sind. Dies ist überwiegend bei dynamischem Systemverhalten der Fall. Bei der Simulation werden Experimente an einem Modell durchgeführt, um Erkenntnisse über das reale System zu gewinnen. Das verwendete Simulationsmodell stellt dazu eine Abstraktion des realen Systems dar und enthält Informationen über dessen Struktur, Funktion und Verhalten. Die aus der Simulation gewonnenen Erkenntnisse können dann interpretiert und auf das reale System übertragen werden. Die Simulation findet unter anderem Anwendung, wenn eine Untersuchung am realen System zu aufwändig, zu teuer, ethisch nicht vertretbar oder zu gefährlich ist, oder das reale System (noch) nicht existiert. /14, 17/

Die Durchführung einer Simulation wird als **Simulationsprozess** bezeichnet. Nach Bungartz lässt sich dieser in folgende Schritte aufteilen (Bild 2-1):

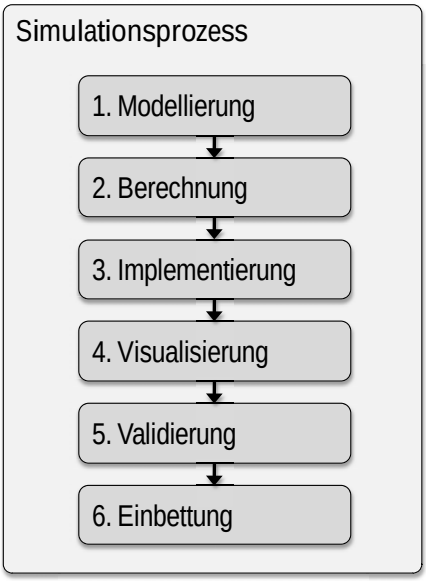


Bild 2-1: Simulationsprozess nach /18/

In der Modellierung wird eine vereinfachende formale Beschreibung des zu untersuchenden Gegenstandes gebildet, die den Ausgangspunkt für die Berechnung bildet. Die Berechnung erfolgt in der Anwendung von Berechnungsvorschriften, in denen Aspekte aus der Realität z.B. in Form von Bewegungsgleichungen oder Differenzialgleichungen mathematisch dargestellt werden. Die Berechnung wird im nächsten Schritt implementiert, d.h. sie wird in ein für den Computer lesbares Format gebracht. Dies kann zum Beispiel mit Hilfe einer textbasierten (z.B. C/C++) oder graphischen (z.B. Matlab/Simulink) Programmiersprache erfolgen. Um eine Bewertung und Interpretation der Simulationsdaten durchführen zu können, ist aufgrund der großen Datenmengen eine geeignete Visualisierung notwendig. Bei der anschließenden Validierung gilt es festzustellen, ob das Ergebnis valide ist oder wo sich Fehlerquellen befinden können, im erstellten Modell, im Code oder in der Berechnung. Die erstellten Simulationen müssen schließlich in geeigneter Weise mit entsprechenden Schnittstellen in den Entwicklungs-/ Produktionsprozess eingebunden werden, um so die Ergebnisse der Simulation sinnvoll für die Entwicklung nutzen zu können.

2.1.3 Mapping und Modelltransformation

Beim Mapping handelt es sich um eine Methode zum Vergleich und zur Herstellung der Beziehung zwischen Modellelementen aus verschiedenen Simulationswerkzeugen. Die Eingänge, Ausgänge und Parameter der Modellelemente von Ausgangs- und Zielwerkzeug werden aufeinander abgebildet.

Das Mapping bildet die Voraussetzung für die Modelltransformation, bei der die zuvor identifizierten Mappings angewandt werden und die Modellelemente des Ausgangswerkzeugs durch die entsprechenden Modellelemente des Zielwerkzeugs ersetzt werden.

Die Methoden zu Mapping und Modelltransformation werden in Kapitel 7.1 und Kapitel 7.2 entwickelt und detailliert beschrieben.

2.1.4 Wiederverwendbarkeit von Modellen

In der Literatur existiert keine einheitliche Definition für den Begriff der "Wiederverwendbarkeit von Simulationsmodellen". Ein Softwaremodul kann als ein Modell betrachtet werden, was die folgende Definition erlaubt:

„Reusability - The degree to which a software module or other work product can be used in more than one computer program or software system." /19/

(deutsch: Wiederverwendbarkeit ist der Grad, in welchem ein Softwaremodul in mehr als einem Computerprogramm oder Softwaresystem genutzt werden kann.)

In der Softwaretechnik bietet das Reifegradmodell nach Rezagholi /20/ eine Einteilung in fünf Stufen als eine Skala für die Wiederverwendung von Softwaremodellen. Mit zunehmendem Reifegrad wird die anwendungs- und anwendungsfeldübergreifende Lösung anstelle der Einzelprogrammierung in den Mittelpunkt der Entwicklung gestellt. /21, 22/

Speziell für Simulationsmodelle existiert keine solche Kategorisierung. Ein Simulationsmodell im Sinne dieser Arbeit unterscheidet sich zum Softwaremodell jedoch lediglich in den verwendeten Beschreibungsformen. Da sich das Reifegradmodell somit auf Simulationsmodelle übertragen lässt, werden die fünf Stufen im Folgenden kurz erläutert.

Stufe 1: (Ungeplante) Ad hoc-Wiederverwendung

Die oftmals ungeplante Wiederverwendung auf Stufe 1 findet in der Regel unsystematisch, unkontrolliert und undokumentiert statt. Dabei wird Software aus vorhandenen Projekten kopiert und an die gegebenen Anforderungen eines neuen Projekts angepasst. Der Nachteil bei einer Wiederverwendung dieser Art ist, dass bei einer notwendigen Änderung an der ursprünglichen Software diese Änderung auch an allen Kopien durchgeführt werden muss. Dies zieht wiederum zusätzlichen Testaufwand nach sich. /21, 22/

Stufe 2: (Ungeplante) Wiederverwendung verfügbarer Softwarekomponenten

Obwohl auch die Wiederverwendung auf Stufe 2 oftmals ungeplant erfolgt, zeichnet sich diese Stufe durch eine strukturierte Vorgehensweise aus, bei der die verfügbare Software genau dokumentiert ist. Durch die Dokumentation wird die Identifizierung der für ein neues Projekt relevanten Softwarekomponenten bereits wesentlich vereinfacht. Dennoch ist der Aufwand für die Wiederverwendung sehr hoch, da bestehende Softwarekomponenten selten in der Form vorhanden sind, in der sie in einer anderen Anwendung benötigt werden. Der notwendige Anpassungsaufwand würde den Wiederverwendungsnutzen übersteigen, so dass der Grad der Wiederverwendung auf dieser Stufe noch sehr gering ist. /21, 22/

Stufe 3: (Geplante) Wiederverwendung innerhalb einer Anwendung

Das entscheidende Merkmal der Wiederverwendung auf Stufe 3 ist, dass Komponenten schon beim Entwurf gezielt für die Wiederverwendung entwickelt werden. Die Wiederverwendung findet geplant statt und kann sowohl innerhalb einer Anwendung als auch projektübergreifend erfolgen. Dazu existiert auf dieser Stufe entweder bereits eine geeignete Infrastruktur für die wiederverwendungsorientierte Softwareentwicklung oder diese wird geschaffen. Teil einer solchen Infrastruktur sind projektübergreifend verfügbare Bibliotheken wiederverwendbarer Komponenten sowie Richtlinien für die Entwicklung, Anpassung und den Transfer von wiederverwendbaren Komponenten. Die Komponentenbibliothek wird auf dieser Stufe noch stark anwendungs- bzw. abteilungsorientiert genutzt. /21, 22/

Stufe 4: Geplante, anwendungsfeldübergreifende Wiederverwendung

Die Wiederverwendung auf Stufe 4 basiert auf einer über alle Anwendungsfelder vereinheitlichten Entwicklungsumgebung mit den dafür notwendigen Verfahren, Methoden und Tools. Anwendungs- und anwendungsfeldübergreifende Architekturmodelle sichern dabei die systematische Wiederverwendung von Komponenten. Durch die Analyse der verschiedenen Anwendungsfelder werden bereits im Vorfeld die Voraussetzungen für eine effektive Wiederverwendung von eigenen oder gekauften Softwarekomponenten geschaffen. Für die Definition, Speicherung und Verwaltung aller für den Softwarelebenszyklus relevanten Informationen von der Planung bis hin zur Wartung und den Zugriff darauf werden Repositories eingerichtet. /21, 22/

Stufe 5: Organisationsweite Wiederverwendung

Während auf Stufe 4 bereits alle technischen Maßnahmen für eine systematische Wiederverwendung vorhanden sind, werden diese in Stufe 5 noch durch die organisatorische und strategische Ausrichtung ergänzt. Das bedeutet, dass auf dieser Stufe die Aktivitäten aller Unternehmensbereiche wie Vertrieb, Marketing, Entwicklung usw. vollständig auf die Wiederverwendung ausgerichtet werden. Alle Aktivitäten, insbesondere die des Verkaufs, werden unter Berücksichtigung verfügbarer Komponenten ausgeübt und nehmen bis hin zu den strategischen Entscheidungen eines Unternehmens ihren Einfluss. Die Mehrheit der Entwicklungen erfolgt auf Basis möglichst existierender Softwarekomponenten. /21, 22/

Im Rahmen dieser Arbeit werden verschiedene bestehende Konzepte untersucht und hinsichtlich ihrer Wiederverwendbarkeit auf den Ebenen Domäne, Disziplin, Werkzeug und Modell bewertet. Die Unterteilung in diese vier Ebenen und deren Zusammenhang ist in Bild 2-2 dargestellt.

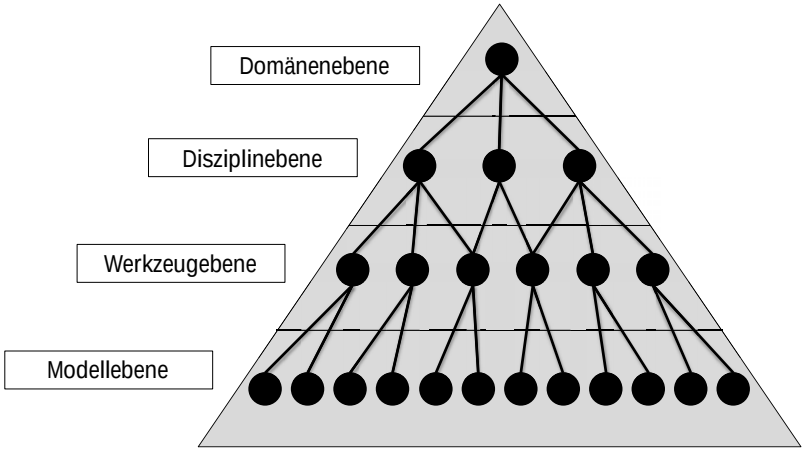


Bild 2-2: Ebenen der Modellwiederverwendung

Zusätzlich werden die einzelnen Konzepte in das Reifegradmodell der Wiederverwendung nach Rezagholi eingeordnet.

2.2 Anforderungen an das Konzept

Im Rahmen dieser Arbeit soll ein Konzept entwickelt werden, das es erlaubt, Simulationsmodelle aus verschiedenen Domänen, Disziplinen, Werkzeugen und Modellen wiederzuverwenden. Die derzeit vorherrschende Inkompatibilität von Simulationsmodellen soll dadurch überwunden und somit die systematische Zusammenarbeit verschiedener Domänen und Disziplinen gefördert werden.

Die wichtigste zu erfüllende Anforderung an das zu entwickelnde Konzept ist dabei die einer **„vollwertigen“ Wiederverwendung**, das heißt, dass ein reiner Zugriff auf ein Simulationsmodell aus einer anderen Domäne, Disziplin oder einem anderen Simulationswerkzeug nicht ausreicht. Hingegen soll ein vollständiger, möglichst automatischer Austausch der im Rahmen der Produktentwicklung benötigten digitalen Daten erreicht werden. Auf diese Weise sollen die bestehenden Systembrüche im Entwicklungsprozess reduziert werden, so dass dieser durchgängiger erfolgen kann. Die Simulationsmodelle sollen domänen- und disziplinübergreifend nutzbar sein und ebenfalls werkzeugübergreifend wiederverwendet werden können. Das wiederverwendete Modell soll jedoch

nicht als Black-Box vorliegen, sondern einsehbar sein und in seinem Systemverhalten veränderbar bleiben.

Die gängigsten Arten der Modellbeschreibung, die **graphikorientierte** und die **sprachorientierte Modellbeschreibung** müssen dabei unterstützt werden. So sollen zum einen rein textuell beschriebene Simulationsmodelle wiederverwendet werden können, zum anderen auch graphisch beschriebene Simulationsmodelle mit graphischer Repräsentation im Blockschaltbild. Diese soll bei der Wiederverwendung erhalten bleiben.

Schließlich wird eine Wiederverwendung der Simulationsmodelle **auf Stufe 4 des Reifegradmodells nach Rezagholi** gefordert, da diese geplant und anwendungsfeldübergreifend mit einer systematischen Wiederverwendung der im Simulationsmodell enthaltenen Komponenten erfolgen soll. Ein vollständiger Zugriff auf alle relevanten Informationen soll stets gewährleistet sein.

Im folgenden Kapitel wird der aktuelle Stand der Forschung und Technik bereits vorhandener Konzepte zur Wiederverwendbarkeit von (Simulations-) Modellen zunächst analysiert und abschließend hinsichtlich der definierten Anforderungen bewertet. Dazu werden die untersuchten Ansätze in eine neu definierte Wiederverwendungsmatrix eingeordnet, in der sie zum einen bezüglich des Reifegrads der Wiederverwendung nach Rezagholi und zum anderen bezüglich ihrer Wiederverwendbarkeit auf Modell-, Werkzeug-, Disziplin- und Domänenebene bewertet werden.

3 Stand der Technik

In diesem Kapitel werden vorhandene Konzepte aus Forschung und industrieller Praxis hinsichtlich der zuvor definierten Anforderungen an wiederverwendbare Simulationsmodelle untersucht und abschließend bewertet.

3.1 Konzepte zur domänenübergreifenden Wiederverwendung

Eine domänenübergreifende Wiederverwendung von (Simulations-)Modellen bedeutet, dass die Verwendung nicht auf eine einzelne Domäne beschränkt ist, sondern sich über mehrere Domänen erstrecken kann. Im Folgenden werden Konzepte untersucht, mit denen eine domänenübergreifende Wiederverwendung ermöglicht werden kann.

3.1.1 Standardisierte Beschreibungsformen für Modelle

Seit 1997 ist **UML** eine standardisierte graphische Sprache zur Modellierung von Software mit Hilfe von geometrischen Elementen. Von der UML werden diese Elemente und deren Bedeutung definiert, wobei der Grundgedanke aller Elemente das Paradigma der Objektorientierung ist. UML stellt ausschließlich eine graphische Notation und keine Methode zur Entwicklung von Software dar. Insgesamt existieren im Rahmen der UML 14 verschiedene Diagrammarten, mit deren Hilfe unterschiedliche Aspekte der Software dargestellt werden können. Grundsätzlich lassen sich mit UML statische, strukturelle sowie auch dynamische Aspekte der Software modellieren. /23/

UML wird in unterschiedlichen Domänen jeweils in der Disziplin Softwaretechnik verwendet. Eine Verbesserung der Wiederverwendbarkeit ergibt sich durch die graphische Darstellung und die große Verbreitung als Standard, der von vielen Entwicklern genutzt wird. UML wurde für die Entwicklung objektorientierter Software entworfen und basiert deshalb auf den objektorientierten Prinzipien, die für die Wiederverwendbarkeit wichtig sind. Die Wiederverwendbarkeit von Modellen, die auf UML basieren, ist im Reifegradmodell in Stufe 2 einzuordnen. UML wird für die Beschreibung von Software und Softwaremodellen bereits in allen Domänen eingesetzt. Typische Beschreibungsformen für Simulationsmodelle werden nicht unterstützt.

SysML basiert auf UML und erweitert diese um Elemente zur Beschreibung von Aspekten des Systems Engineering. Im Gegensatz zur UML, die auf die Softwareentwicklung

fokussiert ist, ist mit Hilfe der SysML eine Modellierung über alle Disziplinen (Mechanik, Elektrik, Softwaretechnik) hinweg möglich. Weiterhin erfolgt die Verknüpfung der Architektur mit der Anforderungsanalyse. Die frühzeitige Erfassung von Anforderungen soll eine Validierung des Gesamtsystems ermöglichen. SysML ist ein phasenübergreifendes Konzept, das unter anderem auch Simulationsdaten in seinem Datenmodell enthält. /24, 25/

Typische Beschreibungsformen für Simulationsmodelle werden jedoch nicht unterstützt. Im Reifegradmodell kann SysML auf Stufe 2 eingeordnet werden.

Im Jahr 2006 wurde das herstellerunabhängige, neutrale Datenformat **AutomationML** (Automation Markup Language) entwickelt, um den Planungsprozess von fertigungs- und prozesstechnischen Anlagen zu verbessern. Dazu sollte ein durchgängiger Datenaustausch geschaffen werden, der auf Grund der Inkompatibilität der im Planungsprozess verwendeten Softwarewerkzeuge bis dahin nicht möglich war. Bei AutomationML handelt es sich ausschließlich um ein Datenformat und keine Methode oder einen Prozess. Das Datenformat ist jedoch kein neues, sondern besteht lediglich in einer Verknüpfung vorhandener, XML-basierter Formate. Folgende drei Formate sind in AutomationML miteinander verknüpft: CAEX (Computer Aided Engineering Exchange) wird für die Modellierung der Anlagentopologie verwendet. COLLADA (COLLAborative Design Activity) ist ein Datenmodell zur Beschreibung geometrischer und kinematischer Zusammenhänge. Schließlich kann mit PLCOpen das steuerungstechnische Verhalten der Maschine abgebildet werden. Da die in AutomationML genutzten Formate auf XML basieren, wird eine plattformunabhängige Modellierung gewährleistet. /26/

AutomationML ermöglicht somit eine domänen- und disziplinübergreifende Wiederverwendung von Daten. Im Reifegradmodell ist AutomationML auf Stufe 3 einzuordnen. Die Wiederverwendbarkeit wird dabei durch die flexible Verknüpfung vorhandener Standards, Erweiterbarkeit und einer Herstellerunabhängigkeit gewährleistet. Eine werkzeugübergreifende Wiederverwendung von Simulationsmodellen mit graphischer Repräsentation ist jedoch nicht möglich.

Modelica ist eine objektorientierte Beschreibungssprache für physikalische Modelle, die seit 1996 von Simulationsexperten aus Europa, den USA und Kanada, die sich zur „Modelica Association“ zusammengeschlossen haben, entwickelt wird. Sie eignet sich zur fachbereichsübergreifenden Beschreibung von Modellen aus der Mechanik, Elektrotech-

nik und Elektronik, Thermodynamik, Hydraulik und Pneumatik, Regelungstechnik und Prozesstechnik. Modelica basiert im Wesentlichen auf der von Hilding Elmqvist objektorientierten Modellierungsmethodik und realisiert einen objektorientierten Ansatz zur Modellierung, dessen Ursprung in der Softwareentwicklung liegt. Objektorientiertheit bedeutet in diesem Fall eine hierarchische Strukturierung der Modelle sowie die Tatsache, dass ein Objekt einer Entität aus der Realität entspricht. Eine Entität ist eine Informationseinheit, die aus einer bestimmten Perspektive als Ganzes gesehen wird. Die graphische Darstellung wurde in Modelica mit Hilfe von Objektdiagrammen realisiert. Damit eine vielfältige Einsetzbarkeit der Modelle gewährleistet ist, können Objekte mit Parametern versehen werden. /27/

Modelica wird Open-Source angeboten und ist somit einem breiten Publikum zugänglich. Beiträge zur Verbesserung der Datenmodellierung werden durch die modular hierarchische Strukturierung geleistet. Eine große Verbreitung, sowie eine breite Unterstützung an mechatronischen Beschreibungsmitteln und eine objektorientierte Informationsbeschreibung erhöhen die Wiederverwendbarkeit der damit modellierten Systeme und Systemteile. Diese Art der Wiederverwendung ist auf Stufe 3 des Reifegradmodells einzuordnen. Mit Modelica wird sowohl die gekoppelte Simulation als auch eine werkzeugübergreifende Nutzung von Modellen möglich. Hierbei wird jedoch vorausgesetzt, dass die Werkzeuge Modelica interpretieren können. Daraus folgt, dass die Wiederverwendung vorhandener Modelle, deren Werkzeuge nicht Modelica-kompatibel sind, nicht möglich ist. Eine Wiederverwendung wird erst nach einer Konvertierung in Modelica möglich. Somit leistet Modelica zwar einen Beitrag zur Verbesserung der Datenmodellierung, die universale Wiederverwendung von Simulationsmodellen auf Disziplin-, Werkzeug- und Modellebene ist jedoch nicht möglich.

Die **MOF** (Meta Object Facility) ist eine Beschreibungsform für Meta-Metamodelle. Meta-Metamodelle enthalten Regeln zur Beschreibung von Metamodellen, die wiederum die Beschreibung konkreter Modelle reglementieren. MOF beinhaltet somit verschiedene Ebenen (Bild 3-1).

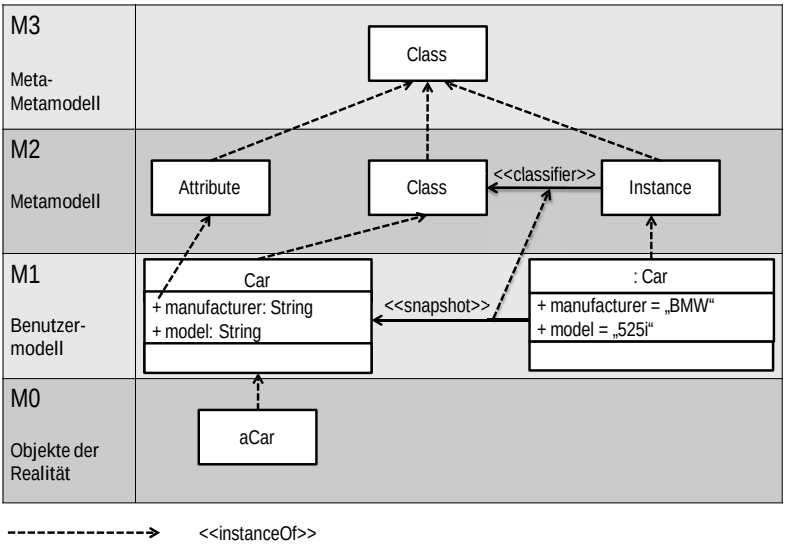


Bild 3-1: Ebenen MOF in Anlehnung an /28/

Auf der untersten Ebene, der M0-Ebene, befinden sich konkrete Objekte aus der Realität. Auf der M1-Ebene erfolgt dann die Modellierung, in der das Objekt formal beschrieben wird, zum Beispiel in der Klasse „Car“ mit den Attributen „manufacturer“ und „model“ vom Typ String. In den Meta-Modellen auf der M2-Ebene erfolgt die Definition, wie die Modelle aufgebaut und strukturiert sind. Dies kann mit Sprachelementen der UML wie Klassen, Assoziationen und Attributen erfolgen. Mit Hilfe der Meta-Meta-Modelle auf der M3-Ebene wird der Aufbau der Metamodelle definiert. Die M3-Ebene bildet die höchste Ebene von MOF. Eine weitere Abstraktion findet nicht statt, um eine unendliche Metaisierung zu vermeiden. /29/

MOF eignet sich zur abstrakten Beschreibung von Simulationssprachen bzw. Datenformaten für Modell-zu-Modell-Transformationen und ist auf Stufe 4 im Reifegradmodell einzuordnen. MOF wird in WieMod zur generischen Beschreibung der Modelle der verschiedenen Simulationswerkzeuge in Form der werkzeugspezifischen Metamodelle und deren Abstraktion, dem WieMod Meta-Metamodell, verwendet.

3.1.2 Prozess- und architekturgetriebene Konzepte

ModelBus ist im Gegensatz zu den vorherigen Konzepten prozess- und architekturgetrieben. In einem Softwareentwicklungsprozess werden eine Vielzahl von Werkzeugen zur Spezifikation, Analyse sowie Codierung eingesetzt. Ein effizienter Entwicklungsprozess setzt voraus, dass die verwendeten Werkzeuge im modellgetriebenen Entwicklungsprozess miteinander vernetzt sind. ModelBus bietet deshalb eine Unterstützung bei der Automatisierung von Entwicklungsaufgaben. Dabei ist es möglich, Aufgaben zu definieren, die automatisch ausgeführt werden. Beim Entwurf großer Modelle mit steigender Komplexität wird es für den Nutzer zunehmend schwieriger, den Überblick über Änderungen zu behalten. Aus diesem Grund bietet ModelBus ein ModelRepository an, das eine Versionierung von Modellen, Auschecken und das Zusammenführen von Modellversionen und -teilen unterstützt. Auch werden die beteiligten Entwickler über Änderungen informiert. Die Entwicklung findet oft in weltweit verteilten Entwicklungsteams und mit unterschiedlichen (Entwicklungs-)Werkzeugen statt. Mit Hilfe von ModelBus können die Werkzeuge miteinander Daten über einen speziellen, für Modelle optimierten, Datenbus austauschen. Zu Werkzeugen wie Eclipse, IBM Rational Software Architect, Sparx Enterprise Architect etc. existieren bereits Anbindungen. /30/

In Bild 3-2 sind die Kommunikationsmuster von Model Bus graphisch dargestellt. Die Werkzeuge sind über einen Adapter an den ModelBus angekoppelt. Über diesen wird es den einzelnen Werkzeugen ermöglicht, auf Modelle anderer Werkzeuge zuzugreifen. So hat beispielsweise Werkzeug A über das Modell-Repository Zugriff auf Modelle von Werkzeug B.

Das hier vorgestellte Konzept eignet sich vor allem für den Datenaustausch, der im modellbasierten Entwicklungsprozess anfällt. ModelBus ist durch die Verbreitung als OpenSource-Software beliebig erweiterbar und basiert auf abstrakten Prinzipien wie MOF. Durch die Möglichkeit einer Anbindung über Adapter an den ModelBus und das Modelrepository lässt sich eine Wiederverwendbarkeit auf den Ebenen Domäne, Disziplin und Werkzeug realisieren. Dabei ist jedoch lediglich ein Zugriff auf die Daten der jeweiligen Modelle möglich, die Modelle selber können im anderen Werkzeug nicht genutzt werden. Die Wiederverwendung mit ModelBus ist auf Stufe 3 des Reifegradmodells einzuordnen. Typische Beschreibungsformen für Simulationsmodelle mit graphischer Repräsentation werden nicht unterstützt.

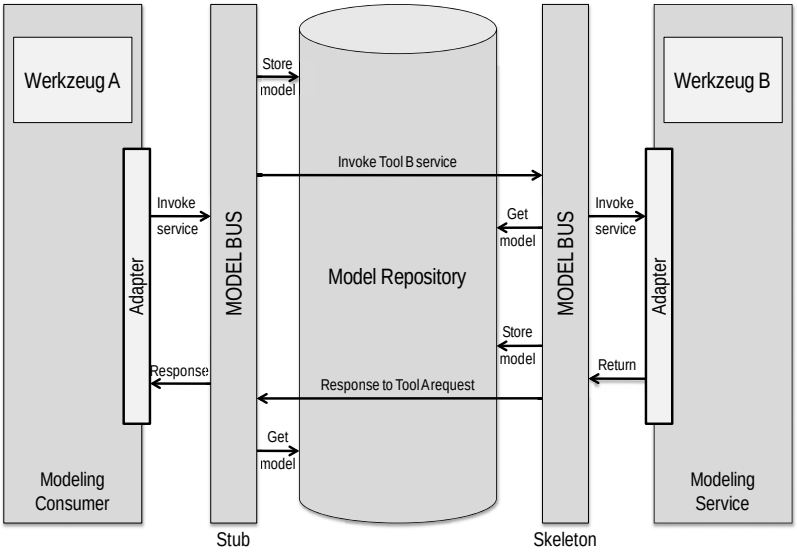


Bild 3-2: Kommunikationsmuster von ModelBus /30/

3.2 Konzepte zur disziplinübergreifenden Wiederverwendung

Die disziplinübergreifende Wiederverwendung von (Simulations-)Modellen beschränkt sich auf jeweils eine Domäne. Innerhalb dieser Domäne können die Modelle jedoch in mehreren Disziplinen wiederverwendet werden. Im Folgenden wird zunächst die älteste und in der Praxis am häufigsten verwendete Engineeringmethode des sequenziellen Engineerings erläutert. Im Anschluss werden verschiedene neuere Engineeringmethoden untersucht, mit denen durch eine disziplinübergreifende Wiederverwendung von (Simulations-)Modellen im Vergleich zum sequenziellen Engineering eine Verbesserung hinsichtlich Kosten, Zeit und Qualität bei der Produktentwicklung erzielt werden soll.

In der Domäne Maschinenbau besteht der vollständige Produktentstehungsprozess aus mehreren Phasen (Bild 3-3 unten). Dabei ist der Engineeringprozess, der die frühen Phasen umfasst, entscheidend für die Kosten des gesamten Produktlebenszyklus. Das Engineering umfasst alle Schritte von Planung und Entwurf bis zur disziplinspezifischen

Entwicklung und Konstruktion, die mit der Erstellung der Konstruktionsunterlagen endet. In der Praxis sind noch überwiegend sequenzielle, disziplinspezifische Engineeringprozesse anzutreffen, in denen die Arbeiten in den Fachdisziplinen Mechanik, Elektrik und Software nacheinander durchgeführt werden (Bild 3-3 mittig). /31/

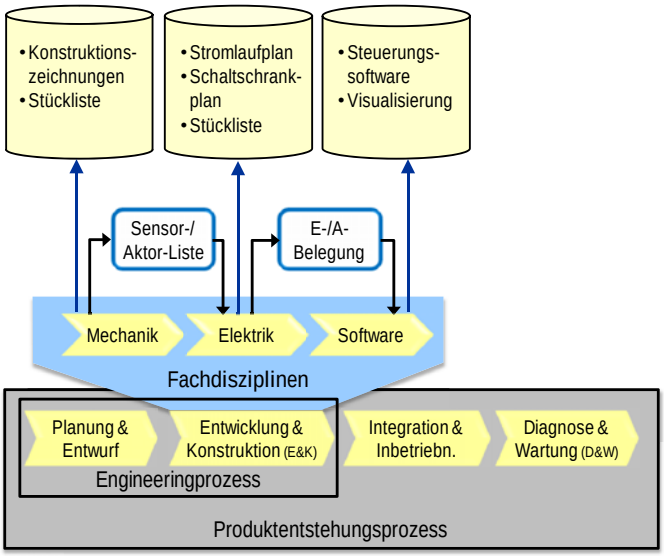


Bild 3-3: Einordnung des sequenziellen Engineeringprozesses in den Produktentstehungsprozess in Anlehnung an /31/

Diese Vorgehensweise erschwert die interdisziplinäre Zusammenarbeit, verlängert die Entwicklungszeit und erhöht dadurch die Kosten für die Produktentwicklung. Die im Folgenden vorgestellten Konzepte verfolgen das Ziel, die interdisziplinäre Zusammenarbeit zu verbessern und gleichzeitig eine disziplinübergreifende Wiederverwendung bestehender Lösungen zu ermöglichen.

3.2.1 Baukastenbasiertes Engineering

Ziel des baukastenbasierten Ansatzes ist, das Engineering durch systematische Wiederverwendung von Komponenten schneller und weniger fehleranfällig zu machen. Dazu muss das Produktspektrum sinnvoll in Komponenten (Bausteine) aufgeteilt und in einem

Baukasten (Bild 3-4 oben) abgelegt werden. Jede Disziplin (Mechanik, Elektrik, Softwaretechnik) verwendet einen eigenen, disziplinspezifischen Baukasten. Aus den Bausteinen kann dann jede auftragsspezifische Variante zusammengestellt, konfiguriert und einsatzspezifisch parametriert werden. Erfordert eine Maschine ein neues Element, so wird dieses entwickelt und als Baustein in den Baukasten aufgenommen. Durch wiederholte Verwendung werden die Bausteine ständig überprüft und verbessert. So wird eine höhere Produktqualität erreicht, die automatisch jedem neuen Projekt zugutekommt. Der Vorteil des baukastenbasierten Engineerings besteht in der automatischen Generierung der Unterlagen für die jeweilige Disziplin (Bild 3-4 unten). Dabei werden Regeln berücksichtigt, die beispielsweise die Kombinationsmöglichkeiten der Bausteine definieren. Die Generierung aus dem Baukasten ist dadurch wesentlich weniger fehleranfällig als ein manuelles Erstellen der Unterlagen. /10, 32/

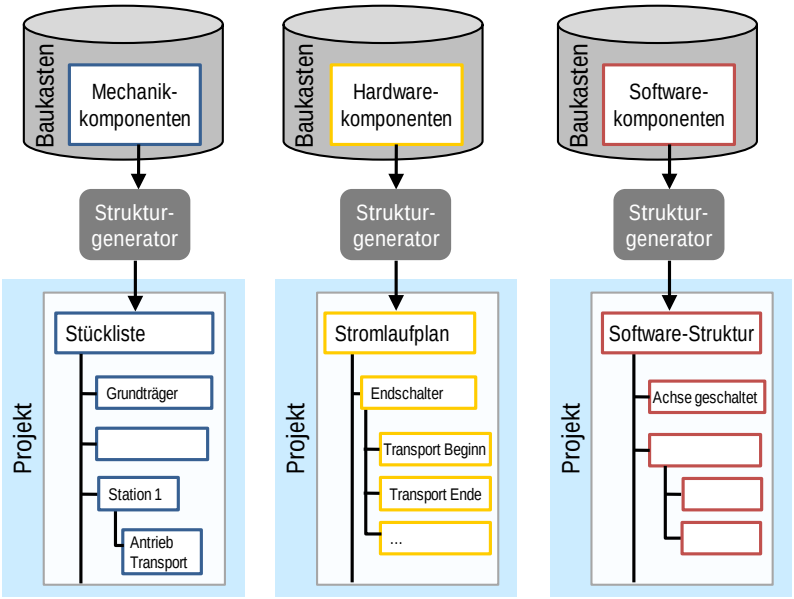


Bild 3-4: Baukastenbasiertes Engineering in Anlehnung an /31/

Durch das baukastenbasierte Engineering wird die Wiederverwendung der im Baukasten abgelegten Komponenten auf Disziplinebene ermöglicht. Es leistet dadurch einen Bei-

trag zur Verbesserung des Engineeringprozesses, kann jedoch keine werkzeugübergreifende Wiederverwendung von Simulationsmodellen ermöglichen.

3.2.2 Mechatronisches Engineering

Beim Konzept des mechatronischen Engineerings arbeiten die Disziplinen Mechanik, Elektrik und Softwaretechnik parallel. In den dabei eingesetzten Baukästen werden mechatronische Komponenten abgelegt, die von den Disziplinen gemeinsam entwickelt werden. Dabei erfolgt die Strukturierung eines Produktes, zum Beispiel einer Maschine, auf Basis funktionaler Einheiten, z.B. Funktion „Heben“. Diese Vorgehensweise wird funktionale Projektierung genannt (Bild 3-5). /33/

Im Rahmen des Projekts **Föderal**, das von 2001 bis 2004 vom Bundesministerium für Bildung und Forschung gefördert wurde, wurden Methoden entwickelt, die eine disziplinübergreifende, baukastenbasierte Projektierung von Maschinen und Anlagen auf Basis vorhandener Engineeringsysteme ermöglichen. Zur Anwendung dieser Methoden wurde das Föderal-Tool entwickelt, das den Aufbau mechatronischer Baukästen unterstützt. Um eine Reduktion der spezifisch für ein Kundenprojekt erforderlichen Engineeringleistungen zu erreichen, erfolgt in der Organisation des Engineerings eine Unterscheidung zwischen projektspezifischen und projektneutralen Tätigkeiten. Die Tätigkeitsschwerpunkte im Engineering liegen zum einen auf der Entwicklung wiederverwendbarer Komponenten, zum anderen auf der Projektierung von Maschinen und Anlagen mit Hilfe dieser wiederverwendbaren Komponenten. Ergebnis der Projektierung sind so genannte Konfigurationen. Projektspezifische Konfigurationen ermöglichen die Generierung von Projektunterlagen, wie Stromlaufpläne, SPS-Programme und Kundendokumentationen. Damit die Generierung möglich wird, müssen Komponenten nach vorgegeben Standards und Regeln entwickelt und in einer firmenspezifischen Baukastensystematik hinterlegt werden. Die in Föderal entwickelten Methoden wurden in einem Produkt, dem EPLAN Engineering Center, kommerziell umgesetzt. /31/

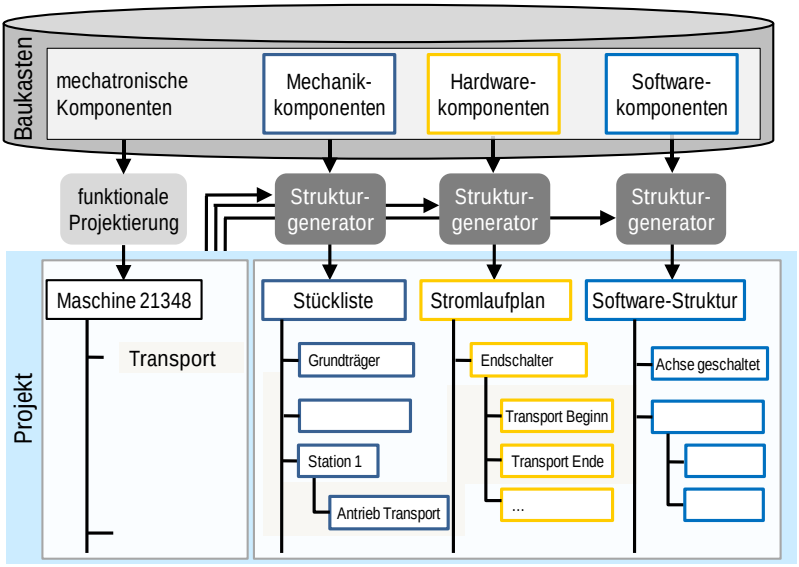


Bild 3-5: Mechatronisches Engineering in Anlehnung an /31/

Föderal unterstützt den Aufbau von Baukästen für das mechatronische Engineering und leistet somit einen Beitrag zur Wiederverwendung von im Baukasten abgelegten mechatronischen Komponenten auf Disziplinebene. Diese Form der Wiederverwendung ist auf Stufe 3 des Reifegradmodells einzuordnen. Die werkzeugübergreifende Wiederverwendbarkeit von Simulationsmodellen wird durch die in Föderal entwickelten Methoden jedoch nicht ermöglicht.

3.2.3 Mechatronischer Entwurf

Beim mechatronischen Entwurf wird unter Einbeziehung aller an der Entwicklung beteiligten Disziplinen definiert, wie das fertige Produkt gestaltet ist und welche Funktionalitäten es haben soll. Die Arbeiten in den einzelnen Disziplinen und die Schnittstellen werden festgelegt, so dass diese Arbeiten zeitgleich erfolgen können. So ist bereits zu einem frühen Zeitpunkt eine bessere Abstimmung möglich, was zu weniger Fehlern und zu erheblichen Zeitvorteilen gegenüber dem sequenziellen Engineering führt (Bild 3-6).

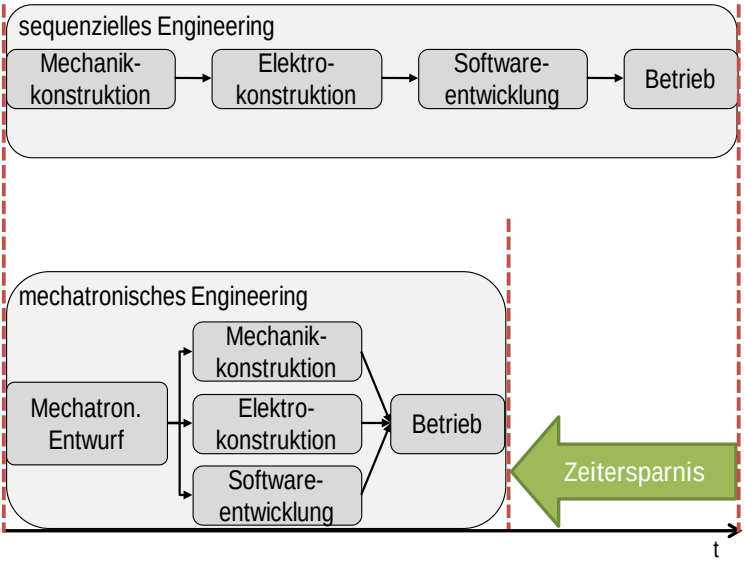


Bild 3-6: Qualitative Darstellung der Zeitersparnis beim mechatronischen Engineering auf Basis eines mechatronischen Entwurfs

Zur Vereinfachung der interdisziplinären Zusammenarbeit beim mechatronischen Entwurf wurde von 2006 bis 2010 das Projekt **AQUIMO** (Adaptierbares Modellierungswerkzeug und Qualifizierungsprogramm für den Aufbau firmenspezifischer mechatronischer Engineeringprozesse) als Nachfolgeprojekt des Projekts Föderal vom Bundesministerium für Bildung und Forschung gefördert. Darin wurden interdisziplinäre, mechatronische Engineering-Prozesse eingeführt. Das im Projekt Föderal entwickelte Baukastensystem diente dabei als eine Grundlage für das im Projekt AQUIMO entwickelte Softwarewerkzeug. Mit Hilfe dieses AQUIMO-Werkzeugs können sitzungsbegleitend mechatronische Entwürfe in verschiedenen Varianten erstellt und bewertet werden. Dabei werden auch Simulationsmodelle generiert, um den Entwurf zu verifizieren. Auf Basis dessen kann anschließend parallel mit der disziplinspezifischen Konstruktion begonnen werden. Um Akzeptanzprobleme zu vermeiden, wurden Einführungsstrategien entwickelt sowie Qualifizierungsmaßnahmen für Mitarbeiter aller an einem Projekt beteiligten Disziplinen durchgeführt. /34/

Das AQUIMO-Werkzeug bietet eine Unterstützung bei der Erstellung und Bewertung mechatronischer Entwürfe. Eine disziplinübergreifende Wiederverwendung von Engineeringdokumenten wird durch die entwickelten Methoden ebenfalls unterstützt. Diese Art der Wiederverwendung ist auf Stufe 3 des Reifegradmodells einzuordnen. Eine werkzeugübergreifende Wiederverwendung von Simulationsmodellen kann durch die entwickelten Methoden jedoch nicht realisiert werden.

3.2.4 Simulationsgestütztes mechatronisches Engineering

Das simulationsgestützte Engineering basiert auf dem baukastenbasierten Engineering. Dabei werden die Baukästen um entsprechende Simulationselemente ergänzt, so dass Simulationsmodelle in gleicher Weise wie Konstruktionsunterlagen automatisch generiert werden können. Aufbauend auf den Konzepten und Methoden des baukastenbasierten Engineerings lässt sich ein simulationsgestützter, mechatronischer Entwicklungsprozess definieren, der auf Basis der Kombination von V-Modell und Quality Gates ein iteratives Vorgehen in mehreren Phasen vorgibt (Bild 3-7). Quality Gates sind definierte Meilensteine, an denen der Konstruktionsfortschritt der einzelnen Fachdisziplinen, die in der Phase des disziplinspezifischen Entwurfs parallel und unabhängig voneinander arbeiten, synchronisiert und abgestimmt wird. Die Abstimmung an den Quality Gates erfolgt mittels aus Baukästen automatisch bedarfs- und entwicklungsstandgerecht generierter Simulationsmodelle, wie es auch bei AQUIMO der Fall ist. Die Simulation dient Anwendern aus unterschiedlichen Fachdisziplinen als gemeinsame Diskussionsgrundlage und trägt wesentlich zum gegenseitigen Verständnis und somit zu einer qualitativ höherwertigen Kommunikation bei. Das iterative Vorgehen führt zu einer schrittweisen Verifikation der einzelnen Entwicklungsschritte. /35, 36/

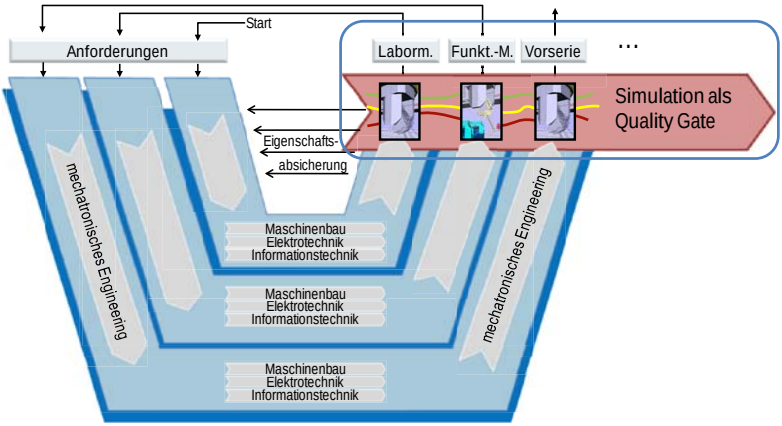


Bild 3-7: V-Modell erweitert um die Simulation als Quality Gate /35/

Das simulationsgestützte Engineering ermöglicht die baukastenbasierte Generierung von Simulationsmodellen. Diese werden an den Quality Gates genutzt, um die Synchronisierung und Abstimmung des Konstruktionsfortschritts der einzelnen Disziplinen zu unterstützen. Somit trägt das simulationsgestützte Engineering zwar zur Verbesserung des Engineeringprozesses bei, die Wiederverwendung der im Baukasten abgelegten Simulationselemente ist jedoch nur auf Disziplinebene möglich. Eine Wiederverwendung auf Domänen-, Werkzeug- und Modellebene wird durch das simulationsgestützte Engineering nicht ermöglicht.

3.3 Konzepte zur werkzeugübergreifenden Wiederverwendung

Bei der werkzeugübergreifenden Wiederverwendung von (Simulations-)Modellen ist die Nutzung der Modelle nicht auf ein einziges (Simulations-)Werkzeug begrenzt, sondern die Modelle können in mehreren Werkzeugen wieder- und weiterverwendet werden.

3.3.1 Konzepte auf Basis eines Standards

STEP (Standard for the Exchange of Product Model Data) ist ein internationaler, in der DIN EN ISO 10303 /37/ festgelegter, Standard, der eine einheitliche Beschreibung für Produktmodelldaten festlegt. Hierzu gehören alle Daten aus dem Produktlebenszyklus wie Produktplanung, Konstruktion, Arbeitsvorbereitung, Herstellung, Nutzung, Recyc-

ling bis zur Entsorgung. Mit Hilfe dieses Standards soll ein durchgängiger Informationsfluss durch Anwendungssysteme wie CAE (Computer Aided Engineering) oder CAP (Computer Aided Planning) ermöglicht werden. Zu diesem Informationsfluss gehören der Produktdatenaustausch, die Produktdatenspeicherung sowie die Produktdatenarchivierung und Produktdatentransformation. Dies wird in STEP realisiert, indem eine einheitliche abstrakte Beschreibung von diesen Daten mit Hilfe der formalen Beschreibungssprache EXPRESS festgelegt wird. /38/

STEP ist ein Standard zur Produktdatenmodellierung. Durch die abstrakte Beschreibung der Produktdaten wird eine Verbesserung der Wiederverwendbarkeit von Datenmodellen sowie des Datenaustausches erreicht. Eine werkzeugübergreifende Wiederverwendung von Simulationsmodellen mit graphischer Repräsentation ist jedoch nicht möglich. Die Art der Wiederverwendung mit Hilfe von STEP ist auf Stufe 2 des Reifegradmodells einzuordnen.

Im Prozessschritt des Anforderungsmanagements ist das Hilfswerkzeug **IntRif** angesiedelt. Heutige Produkte haben meist komplexe Anforderungen mit vielen Abhängigkeiten. An dem Entwicklungsprozess eines Produktes sind häufig viele unterschiedliche Firmen beteiligt, wie die Zulieferer für Automobilhersteller. Jeder Hersteller erfasst die Anforderungen in eigenen Anforderungsmanagement-Tools, welche aber mit anderen Entwicklungspartnern ausgetauscht werden müssen. Bei diesem Austausch gehen jedoch Informationen aus dem Anforderungsmanagement-Tool verloren. Um diesem Problem entgegenzuwirken, wurde von 2008 bis 2010 ein gemeinsames Datenaustauschformat namens RIF (Requirement Interchange Format) von der HIS (Hersteller Initiative Software) entwickelt und anschließend zur Standardisierung an die ProStep iVip Association weitergegeben. Kern dieses Projektes ist es, den Datenaustausch zwischen den Anforderungserfassungswerkzeugen von unterschiedlichen Herstellern zu erleichtern. Das Austauschformat basiert auf einem Metamodell. Dieses gibt vor, wie das individuelle Format für den Austausch zwischen unterschiedlichen Werkzeugen und Unternehmen auszusehen hat, da die auszutauschenden Informationen bei jedem Werkzeug bzw. Unternehmen anders sind. Die kleinste Informationseinheit in diesem Metamodell ist das „Information Element“, dass mit Hilfe der Mechanismen Aggregation, Assoziation und Generalisierung zu größeren Strukturen modelliert werden kann. /39, 40/

Durch das Datenformat entsteht eine Unabhängigkeit von Werkzeugen und durch die Nutzung von Dateien als Informationsträger eine Unabhängigkeit von Datenbanken. Mit

IntRif wird so eine werkzeugübergreifende Wiederverwendung ermöglicht, jedoch werden keine Simulationsdaten, sondern Produktanforderungen ausgetauscht. Diese Art der Wiederverwendung kann auf Stufe 4 des Reifegradmodells eingeordnet werden.

Um den dezentralen Engineeringprozess in der Verfahrens- und Leittechnik zu unterstützen, wurde von BASF, Linde und der RWTH Aachen das neutrale Datenformat **CAEX** (Computer Aided Engineering Exchange) auf Basis von XML entwickelt. Dabei handelt es sich um ein neutrales Datenformat, mit dem statische Objekthierarchien modelliert werden können. Beim dezentralen Engineeringprozess sind weltweit verteilt Ingenieure mit unterschiedlichen Werkzeugen am Entwicklungsprozess beteiligt. CAEX ermöglicht den Informationsfluss zwischen den verschiedenen Engineeringwerkzeugen. Nicht nur die Modellierung von verfahrens- sondern auch von fertigungstechnischen Anlagen ist damit möglich. Eine Eignung zur Beschreibung dynamischer Systeme ist durch die statische Verknüpfung von Objekten nicht möglich. Aus diesem Grund wird dieser Typ von Datenmodell auch als Strukturmodell bezeichnet. Eine Abbildung der realen physikalischen Entitäten in Objekte, die Prinzipien Vererbung und Aggregation, sowie Verwendung von Klassen und dem Bibliothekskonzept führen zu einer höheren Wiederverwendbarkeit. /41/

Mit diesem neutralen Datenformat wird ein Beitrag zur verbesserten Wiederverwendbarkeit in der Datenmodellierung geleistet. Hauptsächlich werden statische und topologische Informationen aufgenommen. Die Beschreibung von Verhalten oder anderen Maschineneigenschaften wird jedoch nicht unterstützt, so dass der Einsatz für Simulationsmodelle im Maschinenbau nicht möglich ist. Die Wiederverwendung mit Hilfe von CAEX ist auf Stufe 3 des Reifegradmodells einzuordnen.

3.3.2 Gekoppelte Simulation

Die Kopplung unterschiedlicher Simulationswerkzeuge ermöglicht die Simulation eines Gesamtsystems, das aus Teilsystemen unterschiedlicher Disziplinen besteht /42/. In der Literatur werden für diese Art der Simulation ebenfalls die Begriffe „Co-Simulation“ und „Simulatorkopplung“ verwendet. Dabei werden Teilsysteme in unterschiedlichen Simulationswerkzeugen, die zur gleichen Zeit arbeiten, simuliert. Die Simulationswerkzeuge tauschen Zustandsgrößen wie beispielsweise Druck und Temperatur oder auch Simulationsparameter wie Zeitschrittweiten und Steuersignale aus. /43/

Bei dieser Art der Wiederverwendung ist lediglich ein Zugriff auf Ergebnisse der Berechnung des jeweiligen Modells in einem speziellen Werkzeug vorgesehen. Eine vollwertige, werkzeugübergreifende Wiederverwendung, d.h. die Nutzung von Modellen aus einem Simulationswerkzeug in einem anderen ist nicht möglich. Eine Möglichkeit zum Datenaustausch wird zwar durch diese Methode erzielt, jedoch setzt sie voraus, dass zwischen allen Simulationswerkzeugen Schnittstellen existieren. Diese Schnittstellen sind für einen universellen Einsatz jedoch nicht geeignet, sondern proprietär für ein konkretes Werkzeug aufgesetzt.

Eine Realisierung der gekoppelten Simulation ist die sogenannte **Simulation Backplane** /44/. Die Aufgabe dieser Backplane ist es, vor der eigentlichen Simulation eine Partitionierung der Daten vorzunehmen und die Berechnungen den Simulationswerkzeugen zuzuordnen. Darüber hinaus übernimmt die Backplane die Synchronisation der einzelnen Simulationswerkzeuge. Das Prinzip der Simulation Backplane ist in Bild 3-8 schematisch dargestellt.

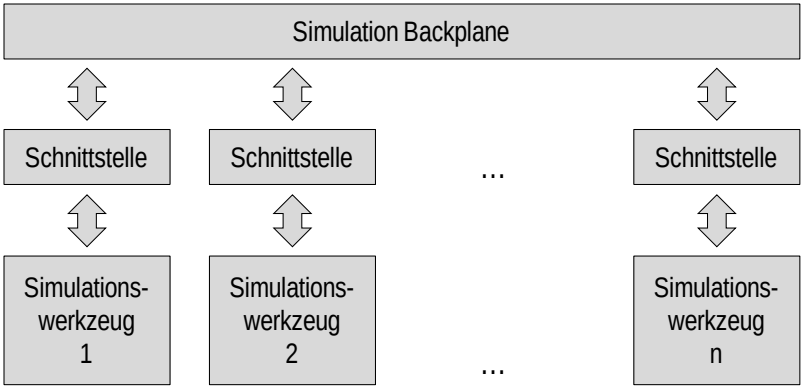


Bild 3-8: Simulation Backplane in Anlehnung an /44/

An die Simulation Backplane können zwar beliebig viele Simulationswerkzeuge gekoppelt werden, jedoch ist wieder lediglich ein Zugriff auf die Ergebnisse der Simulation eines Modells in einem speziellen Werkzeug möglich. Eine vollwertige, werkzeugübergreifende Wiederverwendung von Simulationsmodellen aus anderen Simulationswerkzeugen ist nicht möglich. Die Wiederverwendung erfolgt auf Stufe 3 des Reifegradmodells.

Bei der **High Level Architecture (HLA)** handelt sich um eine weitere Art der gekoppelten Simulation. Sie wurde vom „US Department of Defense“ als Softwarearchitektur für verteilte Softwareumgebungen entworfen und ist seit 2000 ein IEEE-Standard /45/. In ihr werden Design-Regeln, funktionale Komponenten und Schnittstellen für Architekturen zur integrierten und verteilten Simulation spezifiziert. Eine einzelne Simulation in einem bestimmten Simulationswerkzeug kann oftmals nicht alle Funktionen bieten, die der Benutzer verlangt. Mit Hilfe der HLA können die Funktionalitäten unterschiedlicher Simulationen verschiedener Simulationswerkzeuge kombiniert werden. Möglich wird dies, indem einzelne Simulationen per Adapter an die sogenannte RTI (Run-Time-Infrastructure) angebunden werden. Die RTI stellt die zentrale Kommunikations- und Steuerzentrale des Simulationsverbundes dar. Ein Werkzeug, welches an die RTI gebunden wird, trägt die Bezeichnung “Federate”. Mehrere zusammengeschaltete Federates ergeben die Gesamtsimulation, die Federation. /45/

Von der HLA werden die sogenannten HLA-Rules sowie OMTs (Object Model Template) definiert. HLA-Rules sind Vorgaben, wie die Federates über die RTI kommunizieren. Das OMT definiert die Teilnehmer an einem Simulationsverbund sowie die Schnittstellen der Federates. Implementierungen für HLA-Schnittstellen liegen in den Sprachen Java, Ada 95, CORBA (Common Object Request Broker Architecture), IDL (Interface Definition Language) und C++ vor. /45/

Durch HLA können zwar beliebig viele Simulationen miteinander gekoppelt werden, jedoch ist wie bei der Simulation Backplane lediglich ein Zugriff auf die Ergebnisse der Simulation eines Modells in einem speziellen Werkzeug möglich. Eine vollwertige, werkzeugübergreifende Wiederverwendung von Simulationsmodellen aus anderen Simulationswerkzeugen ist nicht möglich. Die Wiederverwendung erfolgt somit ebenfalls auf Stufe 3 des Reifegradmodells.

3.3.3 Proprietäre Lösungen

Um eine werkzeugübergreifende Wiederverwendung einzelner Modellelemente zu ermöglichen, existieren für einige Simulationswerkzeuge proprietäre Speziallösungen. Als Beispiel für eine solche Lösung kann der **MMI-Block** (Matlab Model Interface) im Simulationswerkzeug Virtuos genannt werden /46/. Dieser Block ermöglicht den Import eines linearisierten Matlab/Simulink-Modells in Virtuos. Im MMI-Block sind die Modellinformationen des Matlab/Simulink-Modells enthalten. Die Ein- und Ausgänge des

MMI-Blocks werden beim Import in Virtuos automatisch aus dem Matlab/Simulink-Modell generiert (Bild 3-9).

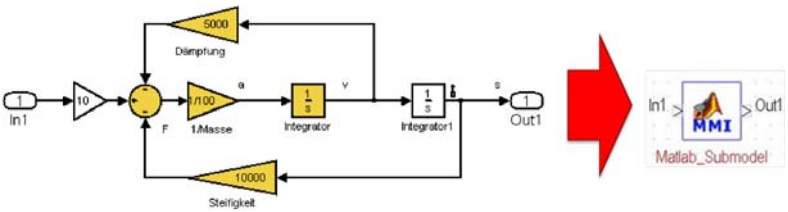


Bild 3-9: Generierung eines MMI-Blocks in Virtuos aus einem Matlab/Simulink-Modell

Diese Lösung birgt jedoch einige Nachteile. So können einerseits nur passive Blöcke, d.h. keine Quellen, importiert werden. Zum anderen ist nur der Import von linearen Modellen möglich. Nicht-lineare Modelle werden beim Import automatisch um einen festzulegenden Arbeitspunkt linearisiert. Das ursprüngliche Matlab/Simulink-Modell ist nicht mehr einsehbar und kann in seinem Systemverhalten nicht mehr verändert werden. Somit ist der MMI-Block lediglich als eine Black-Box, jedoch nicht als vollwertige Wiederverwendung eines bestehenden Modells zu verstehen. /9/ Die Wiederverwendung von Modellen mit dem MMI-Block kann auf Stufe 3 des Reifegradmodells eingeordnet werden.

3.4 Konzepte zur modellübergreifenden Wiederverwendung

Bei der modellübergreifenden Wiederverwendung von (Simulations-)Modellen werden einzelne Modellkomponenten wiederverwendet. Dies geschieht in der Regel innerhalb eines einzelnen (Simulations-)Werkzeugs mit Hilfe der Verwendung von Modellbaukästen und Modellbibliotheken.

Diese Art der Wiederverwendung wird in den Forschungsarbeiten von Lingxiang Xu untersucht mit dem Ziel, die Wiederverwendbarkeit von Simulationsmodellen durch die Art der Erstellung zu erhöhen. Schwerpunkt sind hierbei Simulationsmodelle für den Steuerungstest, um eine Inbetriebnahme von Maschinen schon parallel zum Entwicklungsprozess im Rahmen einer virtuellen Inbetriebnahme durchführen zu können. /47/

Die Modellerstellung erfolgt durch die Nutzung einer Baukastensystematik, die es erlaubt, Komponenten zu entwerfen und in verschiedenen Projekten einzusetzen. Bei der Wiederverwendung einzelner Komponenten in einem konkreten Projekt, müssen diese nur noch entsprechend parametriert werden. Mit Hilfe eines Modulbaukastens wird die einfache Bildung von Maschinenvarianten ermöglicht, was vor allen Dingen für den Sondermaschinenbau relevant ist. /47/

Das entwickelte Konzept zur Steigerung der Wiederverwendbarkeit basiert auf Prinzipien der Softwaretechnik, wie der hierarchischen Strukturierung von Modelldaten, der Objektorientierung und der Parametrierung. Diese Art der Wiederverwendung kann mittels der Methoden des baukastenbasierten Engineerings (Kapitel 3.2.1) ebenfalls realisiert werden.

Die analysierten Konzepte zur modellübergreifenden Wiederverwendung können auf Stufe 3 des Reifegradmodells eingeordnet werden. Die Wiederverwendung kann jedoch nur innerhalb eines einzelnen Simulationswerkzeugs erfolgen, für das die im Baukasten abgelegten Komponenten speziell entwickelt wurden. Eine werkzeugübergreifende Wiederverwendung von Simulationsmodellen kann weder durch das Konzept von Lingxiang Xu noch durch das Konzept des baukastenbasierten Engineerings erzielt werden.

3.5 Zusammenfassung und Defizite

Die untersuchten wissenschaftlichen Konzepte bzw. industriellen Standards und Umsetzungen werden in Tabelle 3-1 kategorisiert. Dazu werden die vertikal aufgetragenen Stufen des Reifegradmodells nach Rezagholi und die horizontal aufgetragenen Ebenen der Wiederverwendung miteinander in Zusammenhang gebracht und zu einer Wiederverwendungsmatrix kombiniert (Tabelle 3-1).

Ebene / Reifegrad	Domänen-übergreifende Wiederverw.	Disziplin-übergreifende Wiederverw.	Werkzeug-übergreifende Wiederverw.	Modell-übergreifende Wiederverw.
Stufe 1				
Stufe 2	UML, SysML	SysML	STEP	
Stufe 3	AutomationML, ModelBus, Modelica	AutomationML, ModelBus, Föderal, AQUIMO	ModelBus, Simulation Backplane, HLA, CAEX, MMI-Block	Xu
Stufe 4	MOF		IntRif	
Stufe 5				

Tabelle 3-1: Einordnung der untersuchten Konzepte in die Wiederverwendungsmatrix

Die Einordnung der untersuchten Konzepte in die Wiederverwendungsmatrix zeigt, dass der Reifegrad der Konzepte auf den Stufen 2 bis 4 einzuordnen ist. Die meisten Konzepte befinden sich auf Stufe 3, das heißt, sie sind für die geplante Wiederverwendung ausgelegt, beschränken sich jedoch auf eine bestimmte Anwendung oder ein bestimmtes Anwendungsfeld. Auf der Stufe der anwendungsfeldübergreifenden Wiederverwendung, Stufe 4, sind lediglich zwei Konzepte eingeordnet (MOF und IntRif). Diese ermöglichen jedoch nur eine Wiederverwendung auf jeweils einer Ebene der Wiederverwendung, nämlich MOF auf Domänenebene und IntRif auf Werkzeugebene. Die Konzepte, die speziell auf die Wiederverwendbarkeit von Simulationsmodellen abzielen, sind ausschließlich auf Stufe 3 eingeordnet.

Die Ziele der untersuchten Konzepte und die identifizierten Defizite sind in Tabelle 3-2 zusammenfassend dargestellt. Dabei wird deutlich, dass zwar Konzepte existieren, die eine Wiederverwendung auf den einzelnen Ebenen (Domäne, Disziplin, Werkzeug und Modell) ermöglichen, jedoch mit keinem der untersuchten Konzepte die Wiederverwendung auf allen Ebenen realisiert werden kann. Bei einigen Konzepten ist eine Wiederverwendung zwar möglich, diese ist jedoch meistens mit hohem Aufwand verbunden, so dass die Wiederverwendung ineffizient ist. Somit kann die Zielsetzung der Wiederverwendbarkeit nur teilweise erfüllt werden. Das Ziel, die Produktentwicklung zu unterstützen, kann durch die untersuchten Konzepte ebenfalls nur unzureichend realisiert werden. Zum Vergleich werden in der letzten Spalte der Tabelle die in dieser Arbeit entwickelten Methoden, die zusammenfassend als „WieMod“ bezeichnet werden, dargestellt. Aus den ermittelten Defiziten werden in Kapitel 4.1 die Problemstellung zusammengefasst und in Kapitel 4.2 die Zielsetzung dieser Arbeit abgeleitet. In Kapitel 5 werden Anforderungen an die Vorgehensweise zur Wiederverwendung von Simulationsmodellen definiert.

	Ziele \ Konzepte	UML	SysML	AutomationML	MOF	ModelBus	Modelica	Föderal	Aquimo	STEP	IntRif	Simulation Backplane	HLA	CAEX	MMI-Block	Xu	WieMod
Wiederverwendbarkeit	Domänenübergreifend	●	●	●	●	●	●	○	○	◑	◑	◑	◑	○	●	◑	●
	Disziplinübergreifend	◑	●	●	○	●	○	●	●	◑	○	◑	○	○	●	◑	●
	Werkzeugübergreifend	◑	◑	○	○	●	○	○	○	●	●	●	●	●	◑	○	●
	Modellübergreifend	◑	○	○	○	○	○	◑	◑	○	◑	◑	○	○	●	◑	●
	Effiziente Wiederverwendung	●	●	◑	●	○	◑	●	●	◑	●	◑	○	◑	●	◑	●
Unterstützung der Produktentwicklung	Frühzeitige Kooperation in der Entwicklung	○	○	○	◑	○	○	●	●	○	○	◑	○	◑	◑	●	●
	Erhöhung von Zuverlässigkeit und Genauigkeit von Simulationsmodellen	○	○	○	◑	○	○	○	○	○	○	◑	○	○	○	○	●
	Effektivere Nutzung von Simulationsmodellen	○	○	○	●	●	◑	◑	◑	○	○	◑	◑	○	◑	◑	●
	Erhöhung der Bedienerfreundlichkeit der Modellwiederverwendung	◑	○	○	○	○	◑	◑	◑	○	○	○	◑	○	●	○	●

erfüllt ● teilweise erfüllt ◑ nicht erfüllt ○

Tabelle 3-2: Übersicht des Standes der Technik

4 Problemstellung, Zielsetzung und Vorgehensweise

4.1 Problemstellung

Auf dem Markt existiert eine Vielzahl verschiedener Simulationswerkzeuge aus unterschiedlichen Domänen und Disziplinen. In diesen Simulationswerkzeugen werden vielfältige Simulationsmodelle erzeugt, die meist inkompatibel zueinander sind. Die Diskrepanz in den Anforderungen der Domänen an ihre Produkte hinsichtlich Kosten, Wartung, Genauigkeit, Umweltbedingungen etc. ist einer der Gründe für die Inkompatibilität der Simulationsmodelle. Die unterschiedlichen Anforderungen der Produkte spiegeln sich somit auch in den Anforderungen für die zugehörigen Simulationsmodelle wider. Bei der Betrachtung der Domänen Maschinenbau und Raumfahrt besteht beispielsweise ein großer Unterschied in den Kosten der Produkte. So sind die Kosten für einen Satelliten z.B. deutlich höher als die für eine Werkzeugmaschine. Zudem ist ein Satellit im Orbit im Gegensatz zu einer Werkzeugmaschine nicht mehr mit vertretbarem Aufwand wartbar. Die Produkte der Raumfahrt erfordern somit eine höhere Genauigkeit, Zuverlässigkeit, Wartungsfreiheit und Langlebigkeit, so dass die Simulation des Betriebs der Produkte bereits sehr genau und zuverlässig sein muss.

Die Engineeringprozesse im Maschinenbau laufen heute noch überwiegend sequenziell ab, d.h. die Konstruktionsarbeiten in den einzelnen Disziplinen werden nacheinander durchgeführt. Die Verifikation der Ergebnisse der Einzeldisziplinen findet meist ohne die anderen Disziplinen statt, was dazu führt, dass disziplinspezifisches Know-How nicht in die Produktentwicklung einfließt. Eine Wiederverwendung vorhandener Simulationsmodelle findet in der Regel nur innerhalb des gleichen Simulationswerkzeugs statt, indem bestehende Modelle aus alten Projekten kopiert und für neue Projekte angepasst und erweitert werden. Die Anpassung und Erweiterung der kopierten Modelle und Modellelemente erfolgt manuell und ist somit zeitaufwändig und fehleranfällig. Ein weiterer Nachteil dieser Vorgehensweise ist, dass auch die Fehler aus alten Modellen kopiert werden und in neuen Modellen erneut Probleme verursachen.

Soll eine werkzeugübergreifende Wiederverwendung von Simulationsmodellen oder einzelnen Elementen daraus erfolgen, d.h. ein Modell oder Modellelemente aus einem spezifischen Simulationswerkzeug sollen in einem anderen Simulationswerkzeug wiederverwendet werden, müssen diese bislang aufwändig manuell nachmodelliert werden. Eine effiziente, werkzeugübergreifende Wiederverwendung von Simulationsmodellen

zur gezielten Nutzung der Stärken unterschiedlicher Simulationswerkzeuge ist somit meist noch nicht möglich, sondern ist mit Doppelarbeit und aufwändiger Datenaufbereitung bei der Modellerstellung verbunden. Daher ist es unter wirtschaftlichen Gesichtspunkten insbesondere für KMU bisher kaum möglich, die Entwicklung komplexer Produkte auf der Grundlage von qualitativ hochwertigen Simulationsmodellen durchzuführen und dabei gleichzeitig den Modellierungsaufwand und die Entwicklungszeiten gering zu halten.

4.2 Zielsetzung

Um die genannten Probleme zu lösen, sollen deshalb in dieser Arbeit ein Konzept für die domänen- und disziplinübergreifende Wiederverwendung von Simulationsmodellen sowie Methoden, um dieses anwenden zu können, entwickelt werden. Da sich die Simulation als anschauliche Diskussionsgrundlage für alle am Produktentwicklungsprozess beteiligten Disziplinen bewährt hat, ist es erstrebenswert, den Einsatz der Simulation zu vereinfachen und wirtschaftlicher zu machen. Eine Wiederverwendung von Simulationsmodellen in den Simulationswerkzeugen der einzelnen Domänen und Disziplinen soll dazu beitragen, indem die dazu bislang notwendige Doppelarbeit und der daraus resultierende Zusatzaufwand bei der Modellerstellung reduziert werden.

Eine Parallelisierung der Arbeitsschritte der einzelnen Disziplinen im Engineeringprozess soll durch die Wiederverwendung von Simulationsmodellen ermöglicht werden, indem disziplinübergreifend ein Simulationsmodell als gemeinsame Diskussionsgrundlage genutzt wird und so als Schnittstelle zwischen den Disziplinen dienen kann. Auf diese Weise wird die Produktentwicklung qualitativ höherwertig und kann gleichzeitig in kürzerer Zeit erfolgen. Die Zeitersparnis resultiert wiederum in einer Kosteneinsparung.

Analog dazu soll auch die domänenübergreifende Zusammenarbeit durch die Wiederverwendung von Simulationsmodellen einfacher und wirtschaftlicher werden. Das Know-How der verschiedenen Domänen, das in domänenspezifischen Simulationsmodellen enthalten ist, soll durch die Möglichkeit der Wiederverwendung auch anderen Domänen zugänglich gemacht werden und deren Simulationsmodellen zugutekommen.

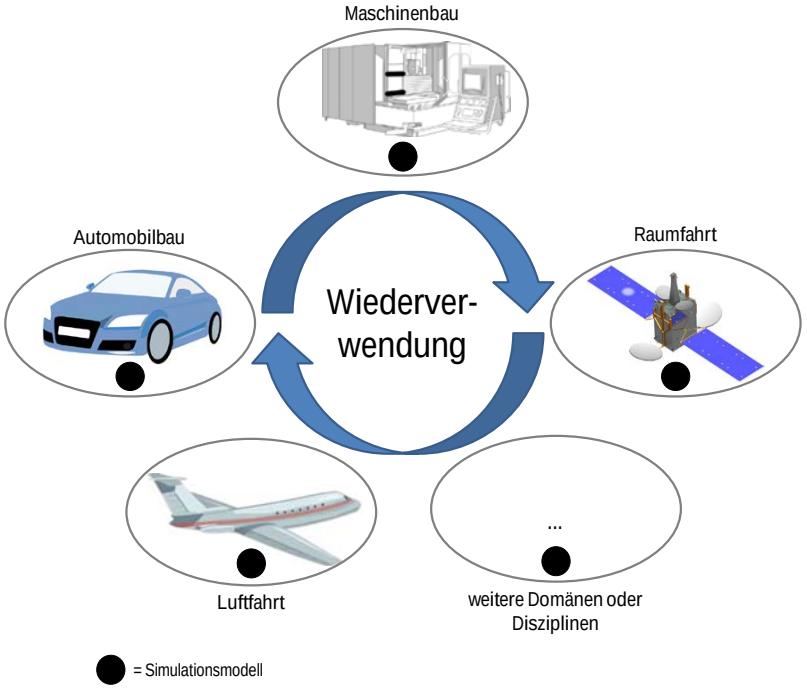
Die Genauigkeit und Zuverlässigkeit der Modelle soll durch die Wiederverwendung maßgeblich erhöht werden, da die Modelle mit unterschiedlichen Simulationswerkzeugen unter verschiedenen Gesichtspunkten getestet werden können. Die Vorzüge und

Stärken der verschiedenen Simulationswerkzeuge können wesentlich effektiver und effizienter genutzt werden, wenn deren Modelle kompatibel zueinander sind. Durch die optimale Kombinationsmöglichkeit verschiedener Simulationswerkzeuge soll eine höherwertige Modellierung erzielt werden.

Mit dem in dieser Arbeit entwickelten Konzept, das zusammenfassend als WieMod bezeichnet wird, soll eine Wiederverwendung auf Stufe 4 des Reifegradmodells ermöglicht werden. In Kapitel 3.5 wurden bereits zwei Konzepte identifiziert, die sich auf Stufe 4 befinden, „MOF“ und „IntRif“. Mit diesen kann jedoch nur eine Wiederverwendung auf jeweils einer Ebene realisiert werden (vgl. Tabelle 3-1). Im Gegensatz dazu erstreckt sich die in dieser Arbeit entwickelte Möglichkeit der Wiederverwendung über alle vier Ebenen (Domänen-, Disziplin-, Werkzeug- und Modellebene) der Wiederverwendungsmatrix.

Um eine solche Wiederverwendung einzelner Modellelemente oder ganzer Modelle in verschiedenen Simulationswerkzeugen unterschiedlicher Domänen und Disziplinen zu erreichen, muss das gewünschte Modell bzw. das Modellelement von der Modellbeschreibungssprache des Ausgangswerkzeugs in die des Zielwerkzeugs transformiert werden. Nach der Transformation soll das Modell oder das Modellelement des Ausgangswerkzeugs wie gewohnt in dem Zielwerkzeug verwendet werden. Dabei ist es wichtig, Informationsverluste bei der Modelltransformation zu vermeiden. Damit sichergestellt ist, dass die Wiederverwendung möglichst effizient erfolgt, werden in dieser Arbeit Kriterien für die Effizienz der Wiederverwendung definiert und das erarbeitete Konzept sowie die entwickelten Methoden für dessen Anwendung bezüglich dieser Kriterien validiert.

Die globale Zielsetzung der domänen- und disziplinübergreifenden Wiederverwendung von Simulationsmodellen ist in Bild 4-1 dargestellt.



Quellen Bilder:
Auto: Auto Vector von MkDesign; Satellit: DLR; Flugzeug: Deutsche Gefaessliga

Bild 4-1: Domänen- und disziplinübergreifende Wiederverwendung von Simulationsmodellen als Zielsetzung

4.3 Vorgehensweise

Die Vorgehensweise zum Erreichen der beschriebenen Zielsetzung ist in Bild 4-2 zusammenfassend dargestellt.

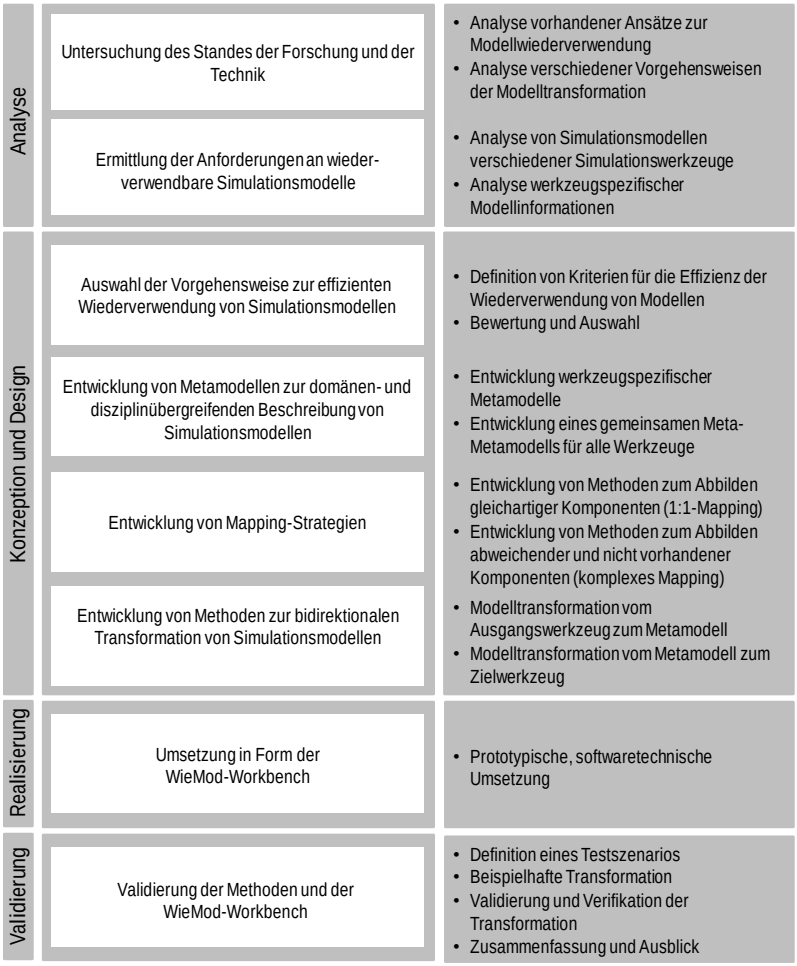


Bild 4-2: Vorgehensweise der Arbeit

Die Vorgehensweise orientiert sich an den Phasen der Softwareentwicklung und ist in die Schritte Analyse, Konzeption und Design, Realisierung und Validierung unterteilt /48/. In den folgenden Abschnitten werden die Arbeiten in den einzelnen Entwicklungsphasen detaillierter erläutert:

Analyse

Basierend auf den in Kapitel 2 definierten, grundlegenden Begriffen und Anforderungen an das zu erarbeitende Konzept für die Wiederverwendung von Simulationsmodellen wurde in Kapitel 3 eine Untersuchung bereits vorhandener Konzepte aus Forschung und industrieller Praxis durchgeführt. In Kapitel 5.1.1 werden Anforderungen an die Vorgehensweise der Wiederverwendung definiert, woraufhin in Kapitel 5.1.2 verschiedene Möglichkeiten zur Modelltransformation analysiert werden. Im Rahmen einer umfangreichen Analyse von Alt- und Fremdmodellen werden in Kapitel 5.2 Anforderungen an wiederverwendbare Modelle ermittelt. In Kapitel 6.1 werden anhand eines einfachen Beispielmodells werkzeugspezifische Modellinformationen für Modelle ausgewählter Simulationswerkzeuge analysiert.

Konzeption und Design

Auf Basis der angestellten Analyse werden Kriterien zur Bewertung der Effizienz bei der Wiederverwendung von Modellen definiert. Auf Grund dieser Kriterien wird in Kapitel 5.1.3 die Transformation über ein Metamodell als die effizienteste Vorgehensweise identifiziert und als Vorgehensweise dieser Arbeit festgelegt. Ein einziges Metamodell erweist sich jedoch im Zuge der Entwicklung als nicht ausreichend, so dass drei verschiedene Metamodelle, eines für jedes prototypisch in das Konzept integrierte Simulationswerkzeug, entwickelt werden, die schließlich in einem übergeordneten Meta-Metamodell abstrahiert werden. Das verwendete Meta-Metamodell wurde im Rahmen der Arbeiten zum Projekt WieMod entwickelt. In Kapitel 6 werden dessen Aufbau, Inhalt und Funktionsweise hergeleitet.

Um die Modellelemente der verschiedenen Simulationswerkzeuge aufeinander abbilden zu können, werden in Kapitel 7.1 so genannte Mapping-Strategien erarbeitet. Dabei handelt es sich um Methoden zur Abbildung gleichartiger bzw. abweichender und nicht vorhandener Modellelemente aus verschiedenen Simulationswerkzeugen. Das Mapping

bildet die Voraussetzung für die Modelltransformation, die in Kapitel 7.2 entwickelt wird.

Realisierung

Die prototypische Realisierung erfolgt in Form der Eclipse-basierten WieMod-Workbench, mit deren Hilfe es ermöglicht wird, Modelle aus verschiedenen Ausgangswerkzeugen zu Modellen für verschiedene Zielwerkzeuge zu transformieren. Die softwaretechnische Umsetzung der WieMod-Workbench wird in Kapitel 8.1 vorgestellt.

Validierung

Zur Validierung des vorgestellten Lösungsansatzes wird in Kapitel 8.2 zunächst ein komplexes Testszenario aus dem Bereich des Werkzeugmaschinenbaus definiert und exemplarisch mit Hilfe der WieMod-Workbench transformiert. Die Transformation wird dann anhand des Originalmodells verifiziert und validiert.

In Kapitel 9 werden die erzielten Ergebnisse zusammengefasst und bewertet. Die umgesetzte Methode sowie der potenzielle Einsatzbereich werden abschließend diskutiert und ein Ausblick auf die Verwendung sowie auf weitere Entwicklungen wird gegeben.

5 Konzeption der Wiederverwendung

In diesem Kapitel wird ein Konzept für die Wiederverwendung von Simulationsmodellen erarbeitet. Die durchgeführten Untersuchungen des Standes der Technik dienen hierzu als Grundlage. In Kapitel 5.1 wird zunächst eine geeignete Vorgehensweise für die Wiederverwendung ausgewählt. In Kapitel 5.2 wird definiert, welche Anforderungen Simulationsmodelle erfüllen müssen, um die gewählte Vorgehensweise anwenden zu können.

5.1 Auswahl der Vorgehensweise

Die Analyse bestehender Konzepte zur Wiederverwendung von (Simulations-) Modellen in Kapitel 3 hat gezeigt, dass ein reiner Zugriff auf die Ergebnisse der Berechnung von Modellen, wie es die gekoppelte Simulation ermöglicht, für eine vollwertige Wiederverwendung auf Stufe 4 des Reifegradmodells nicht ausreicht. Aus diesem Grund wird eine Transformation von Modellen angestrebt, die es erlaubt, ein Modell aus einem Ausgangswerkzeug in ein Modell für ein gewünschtes Zielwerkzeug zu transformieren. Das wiederverwendete Modell soll jedoch nicht als Black-Box vorliegen, gemäß der in Kapitel 2.2 definierten Anforderungen einsehbar sein und in seinem Systemverhalten veränderbar bleiben.

In Kapitel 5.1.1 werden zunächst Anforderungen an die Vorgehensweise der Wiederverwendung, die in Form einer Modelltransformation erfolgen soll, definiert. Daraufhin werden in Kapitel 5.1.2 verschiedene Möglichkeiten zur Modelltransformation analysiert. In Kapitel 5.1.3 werden die gewonnenen Erkenntnisse bewertet und eine Auswahl getroffen.

5.1.1 Anforderungen an die Vorgehensweise

Die Vorgehensweise bei der Wiederverwendung soll möglichst effizient sein, das heißt das Verhältnis von Nutzen und Aufwand soll möglichst groß sein.

$$Effizienz = \frac{Nutzen}{Aufwand}$$

Dieses Verhältnis lässt sich durch die Relation zwischen der Anzahl der Transformationen *m* und Anzahl der Simulationswerkzeuge *n*, in denen die Wiederverwendung stattfinden kann, quantifizieren. Dieser Zusammenhang kann als Wirkungsgrad η der Vorgehensweise betrachtet werden und wird als Kennzahl für die Effizienz der Wiederverwendung eingeführt. Der Wirkungsgrad steigt, wenn zum einen die Anzahl der Simulationswerkzeuge, für die Modelle erzeugt werden können, möglichst hoch ist. Zum anderen muss gleichzeitig die Anzahl der notwendigen Transformationen möglichst klein sein.

$$\eta = \frac{Anzahl\_Simulationswerkzeuge}{Anzahl\_Transformationen} = \frac{n}{m}$$

Zudem soll die Vorgehensweise flexibel erweiterbar sein. Das heißt, die Möglichkeit, neue Simulationswerkzeuge zu den bestehenden hinzuzufügen, muss vorhanden sein. Die neu hinzugefügten Werkzeuge sollen immer sowohl als Ausgangswerkzeug als auch als Zielwerkzeug fungieren können. Das heißt, dass für jedes Simulationswerkzeug eine Eingangs- und Ausgangsschnittstelle verfügbar sein soll. So können zum einen in jedes Simulationswerkzeug fremde Modelle aus anderen Simulationswerkzeugen importiert und somit wiederverwendet werden. Andererseits sollen aus jedem Simulationswerkzeug Modelle derartig transformiert werden, dass sie für andere Simulationswerkzeuge wiederverwendbar sind. Das Hinzufügen neuer Simulationswerkzeuge muss ebenfalls möglichst effizient sein, so dass die Erweiterung mit möglichst geringem Auswand erfolgen kann.

5.1.2 Analyse möglicher Vorgehensweisen

Für die Wiederverwendung von Modellen mit Hilfe einer Modelltransformation existieren prinzipiell zwei alternative Vorgehensweisen, die im Folgenden erläutert werden.

5.1.2.1 Vorgehensweise 1: Direkte Modelltransformation

Die erste Vorgehensweise ist die direkte Modelltransformation von einem Simulationswerkzeug in ein anderes. In Bild 5-1 ist die Vorgehensweise schematisch für *n=4* Simulationswerkzeuge dargestellt. Jeder Doppelpfeil steht für 2 Transformationen. Zwischen 2 Simulationswerkzeugen müssen somit immer 2 Transformationen vorgenommen werden – eine von jedem der beiden Werkzeuge zu dem jeweils anderen. Bei dieser Vorge-

hensweise muss also unabhängig von der Anzahl der Simulationswerkzeuge insgesamt von jedem Werkzeug zu jedem anderen Werkzeug eine Transformation durchgeführt werden. Im Bild sind 6 Doppelpfeile zu sehen, das heißt für *n=4* Simulationswerkzeuge sind *m=12* Transformationen notwendig.

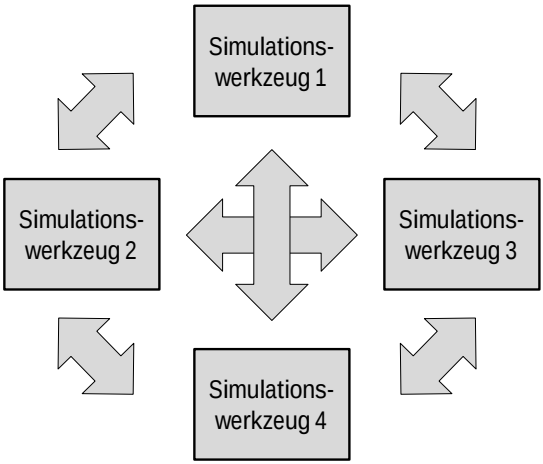


Bild 5-1: Direkte Modelltransformation zwischen Simulationswerkzeugen

Die Formel, die diesen Zusammenhang beschreibt lautet $m=n\cdot(n-1)$. Bei dieser Vorgehensweise werden also für n Simulationswerkzeuge $m=n\cdot(n-1)$ Transformationsalgorithmen benötigt. Für das Beispiel von $n=4$ ergeben sich somit $m=4\cdot(4-1)=4\cdot3=12$ Transformationen.

5.1.2.2 Vorgehensweise 2: Indirekte Modelltransformation über Metamodell

Die zweite Vorgehensweise ist die indirekte Transformation über ein zwischengeschaltetes Metamodell, wie sie in Bild 5-2 dargestellt ist. Zwei der in Kapitel 2 untersuchten Konzepte (MOF, IntRif) nutzen bereits erfolgreich den Metamodellansatz, um eine anwendungsfeldübergreifende Wiederverwendung zu ermöglichen. Bei dieser Vorgehensweise findet keine direkte Transformation von einem Simulationswerkzeug in ein anderes statt, sondern das Simulationsmodell wird zunächst in ein neutrales Datenformat transformiert, in ein so genanntes Metamodell. Das dazu benötigte Metamodell muss zum einen so allgemeingültig sein, dass es die Modelle der verschiedenen Simulations-

werkzeuge beschreiben kann und zum anderen noch so speziell sein, dass keine spezifischen Informationen verloren gehen. Aus einem Ausgangswerkzeug werden die Modellinformationen zunächst in das Metamodell transformiert. Diese Modellinformationen werden dann aus dem Metamodell wiederum in Simulationsmodelle für das spezifische Zielwerkzeug transformiert.

In Bild 5-2 sind wieder die Transformationen für $n=4$ Simulationswerkzeuge zu sehen. Jeder der Pfeile steht für eine Transformation, insgesamt sind 8 Pfeile zu sehen, da für jedes Simulationswerkzeug $m=2$ Transformationen benötigt werden - eine vom Ausgangswerkzeug zum Metamodell und eine vom Metamodell zum Zielwerkzeug.

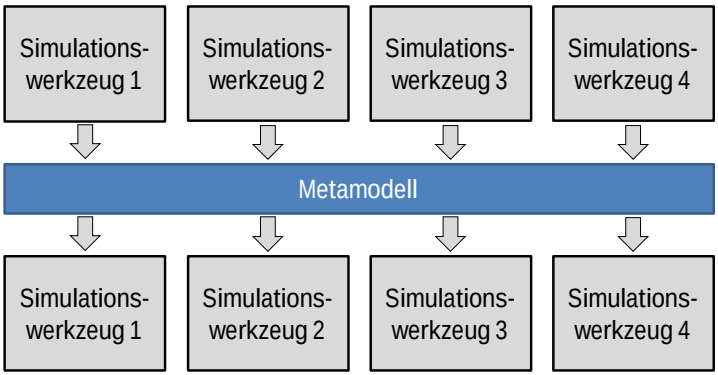


Bild 5-2: Indirekte Modelltransformation über ein Metamodell

Dieser Zusammenhang kann mit der Formel $m=2 \cdot n$ abgebildet werden. Für das Beispiel für $n=4$ ergeben sich somit $m=2 \cdot 4=8$ Transformationen.

5.1.3 Bewertung und Auswahl

Die Untersuchung der beiden alternativen Vorgehensweisen bei der Modelltransformation für $n=4$ Simulationswerkzeuge lässt bereits auf Grund der schematischen Darstellung erste Erkenntnisse über die Effizienz dieser beiden Vorgehensweisen zu. Während Vorgehensweise 1 $m= 12$ Transformationen erfordert, müssen bei Vorgehensweise 2 lediglich $m=8$ Transformationen durchgeführt werden.

Zur Bewertung der jeweiligen Effizienz der möglichen Vorgehensweisen wird der in Kapitel 5.1.1 definierte Wirkungsgrad als quantitatives Maß und Kriterium für die Effizienz der Wiederverwendung verwendet. Für Vorgehensweise 1 ergibt sich ein Wirkungsgrad von $\eta_{Vorgehensweise1} = 0{,}333$.

Hingegen ergibt sich für Vorgehensweise 2 ein Wirkungsgrad von $\eta_{Vorgehensweise2} = 0{,}5$.

Bei der geringen Anzahl an Simulationswerkzeugen ist bereits zu sehen, dass Vorgehensweise 2 effizienter ist. Das volle Potenzial wird jedoch erst bei der Betrachtung der Vorgehensweisen bei wesentlich mehr Simulationswerkzeugen ersichtlich.

Bei der Verwendung von 10 Simulationswerkzeugen sind bei Anwendung der Vorgehensweise 1 bereits *m= 90* Transformationen notwendig, um Modelle von einem Simulationswerkzeug in Modelle für jedes andere überführen zu können. Dies entspricht einem Wirkungsgrad von $\eta_{Vorgehensweise1} = 0{,}111$.

Hingegen sind bei Vorgehensweise 2 lediglich *m =20* Transformationen notwendig, um ein Modell von jedem der 10 Ausgangswerkzeuge in jedem der 10 Zielwerkzeuge verwenden zu können. Dies entspricht wieder einem Wirkungsgrad von $\eta_{Vorgehensweise2} = 0{,}5$.

In Bild 5-3 ist für jede der beiden Vorgehensweisen die Anzahl der jeweils benötigten Transformationen über der Anzahl der Simulationswerkzeuge aufgetragen. Direkt ersichtlich wird, dass sich Vorgehensweise 2 bereits bei *n>3* verwendeten Werkzeugen rentiert.

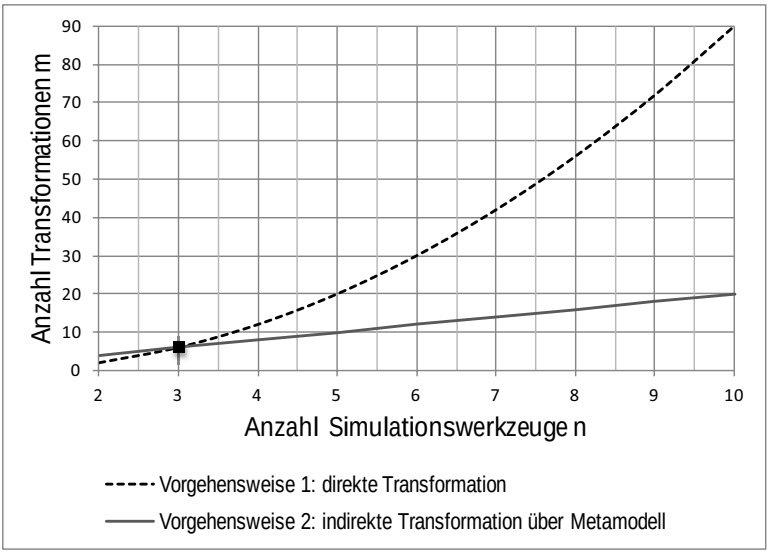


Bild 5-3: Anzahl Transformationen bei direkter und indirekter Transformation

Sollen nun weitere Simulationswerkzeuge zu bereits bestehenden hinzugefügt werden, bestehen auch bei beiden Vorgehensweisen Unterschiede bezüglich des nötigen Aufwands. Bei einer direkten Transformation muss ein Transformationsalgorithmus von jedem Werkzeug in jedes andere und zurück hinterlegt sein, d.h. beim Hinzufügen nur eines neuen Werkzeugs müssen $(n+1) \cdot ((n+1)-1) - n \cdot (n-1) = 2 \cdot n$ neue Transformationsalgorithmen implementiert werden. Der Aufwand einer Transformation über ein Metamodell ist hier deutlich geringer, da beim Hinzufügen eines neuen Simulationswerkzeugs lediglich zwei neue Transformationsalgorithmen hinzugefügt werden müssen, nämlich einer von dem neuen Simulationswerkzeug zum Metamodell und einer vom Metamodell zum Simulationswerkzeug.

Vorgehensweise 1 hat bei $n=2$ Simulationswerkzeugen den optimalen Wirkungsgrad von $\eta=1$. Dieser sinkt jedoch ab der Verwendung von $n=3$ Simulationswerkzeugen mit jedem weiter hinzugefügten. Bei Vorgehensweise 2 hingegen beträgt der Wirkungsgrad unabhängig von der Anzahl der Simulationswerkzeuge immer konstant $\eta=0,5$. (Bild 5-4)

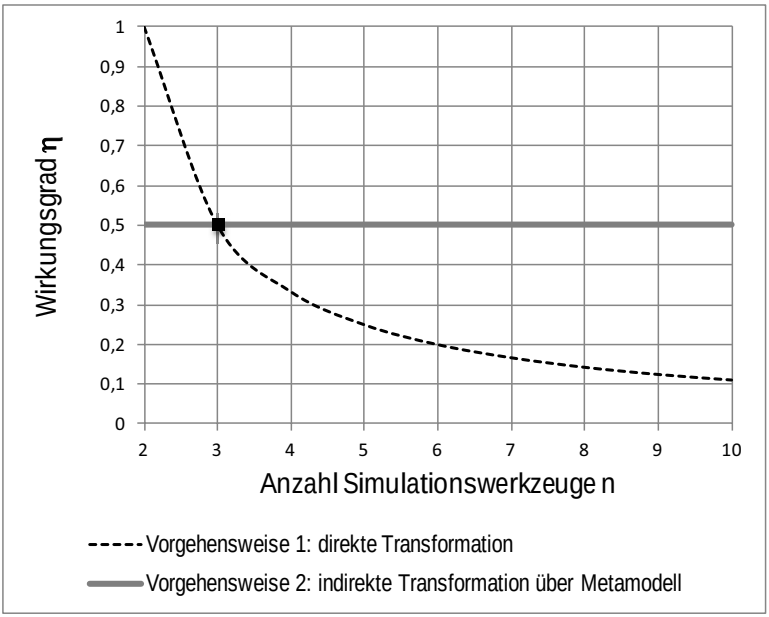


Bild 5-4: Wirkungsgrad bei direkter und indirekter Transformation

Auf Grund der angestellten Untersuchungen erfolgt die Wiederverwendung im Rahmen dieser Arbeit nach Vorgehensweise 2 über ein Metamodell zwischen Ausgangs- und Zielwerkzeug, um die in der Zielsetzung angestrebte Wiederverwendung zu ermöglichen (Bild 5-5).

Nach der erfolgten Auswahl der Vorgehensweise bei der Modelltransformation werden im folgenden Kapitel 5.2 Anforderungen definiert, die wiederverwendbare Simulationsmodelle erfüllen müssen, damit die gewählte Vorgehensweise angewandt werden kann.

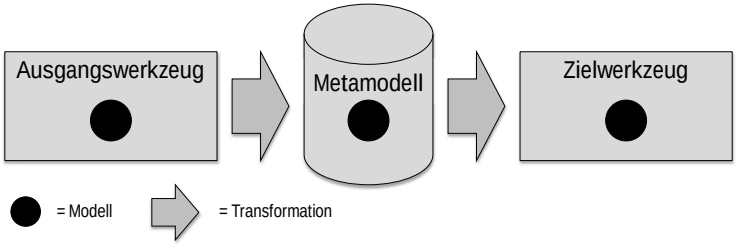


Bild 5-5: Gewählte Vorgehensweise bei der Wiederverwendung

5.2 Ermittlung von Anforderungen an wiederverwendbare Modelle

Zur Entwicklung eines Konzeptes für die Wiederverwendung von Simulationsmodellen wurde zunächst eine umfangreiche Modellanalyse vorhandener Simulationsmodelle verschiedener Simulationswerkzeuge aus mehreren Domänen vorgenommen. Konkret wurden Modelle aus dem Maschinen- und Sondermaschinenbau, dem Anlagenbau sowie der Luft-und Raumfahrt in den dort etablierten Simulationswerkzeugen Virtuos, Matlab/Simulink und SMP2 untersucht. Darin wurde ermittelt, welche Modellelemente der unterschiedlichen Simulationswerkzeuge für die Modellierung von Applikationen für verschiedene Domänen essenziell sind und somit zwingend transformierbar sein müssen.

Ein wesentlicher Unterschied, der zwischen den untersuchten Modellen identifiziert wurde, ist, dass Modelle der Simulationswerkzeuge Virtuos und Matlab/Simulink blockschaltbildbasiert sind, während die Modelle des Simulationswerkzeugs SMP2 rein textuell beschrieben sind. Bei blockschaltbildbasierten Modellen werden die verwendeten Modellelemente durch einzelne Modellbausteine (Blöcke) graphisch repräsentiert. Durch die Verbindung verschiedener Blöcke wird das spezifische Verhalten einer bestimmten Applikation abgebildet. Blockschaltbildbasierte Modelle enthalten zusätzlich zu den reinen Modellinformationen noch Geometrieinformationen, in denen die graphische Repräsentation des Blockschaltbilds gespeichert ist. Rein textuell beschriebene Modellelemente enthalten keine zusätzlichen Geometrieinformationen. Der Umfang der analysierten Virtuos- und Matlab/Simulink-Modelle reichte von einem einfachen Logikmodell bis hin zu einem komplexen Dynamikmodell, das in mehrere Submodelle unterteilt ist.

Bei der durchgeführten Modellanalyse konnten zwei Kategorien von Modellelementen identifiziert werden, Standardelemente und Spezialelemente. Die **Standardelemente**

(Logik, mathematische Operatoren etc.) sind in allen untersuchten Simulationswerkzeugen mit der gleichen Funktionalität vorhanden.

Bei den **Spezialelementen** handelt es sich um Modellelemente mit anwendungsfallspezifischer Funktionalität, die nicht in allen Simulationswerkzeugen vorhanden sind. Diese sind in speziellen Bibliotheken, wie beispielsweise Kollisions-, Materialfluss- oder Mehrkörpersimulations-Bibliothek, auch Toolboxen genannt, vorhanden.

Die durchgeführte Modellanalyse führte zu dem Ergebnis, dass der Großteil der in der Praxis verwendeten Modelle aus Standardelementen aufgebaut ist, die in allen untersuchten Simulationswerkzeugen vorhanden sind. Zudem wurden bei der Modellanalyse die wesentlichen, unverzichtbaren Eigenschaften der untersuchten Modelle identifiziert. Daraus wurden Anforderungen an wiederverwendbare Modelle ermittelt, die erfüllt sein müssen, um die Zielsetzung dieser Arbeit realisieren zu können. Diese identifizierten, grundlegenden Anforderungen sind:

- Lesbarkeit
- Rechenbarkeit
- Wiedererkennbarkeit
- Konfigurierbarkeit

Diese werden in den nachfolgenden Unterkapiteln näher erläutert.

5.2.1 Lesbarkeit

Damit ein Modell aus einem beliebigen Ausgangswerkzeug in einem anderen Zielwerkzeug wiederverwendet werden kann, muss die Anforderung der Lesbarkeit erfüllt sein. Das bedeutet, dass das Dateiformat, in dem das Modell abgespeichert ist, zum einen offengelegt, das heißt für den Anwender zugänglich, und zum anderen interpretierbar ist. Dabei ist es in erster Konsequenz gleichgültig, ob es sich bei der Datei um ein klartextbasiertes Format oder um eine Binärdatei handelt. Grundsätzlich bietet die Verwendung klartextbasierter Dateiformate jedoch wesentliche Vorteile gegenüber Binärdateien. So sind sämtliche im Klartext geschriebenen Konfigurations- und Strukturinformationen für den Menschen lesbar und somit auch manuell änderbar. Zudem lassen sich klartextbasierte Dateien auf beliebigen Computersystemen in einem einfachen Texteditor öffnen und bearbeiten. Schließlich ist der Austausch von Dokumenten mit anderen Anwendun-

gen, neuen Versionen einer Anwendung oder Programmiersprachen erheblich einfacher als bei Binärformaten, da der Inhalt direkt interpretiert werden kann. Beim Binärformat muss zusätzlich die Spezifikation bekannt und vom Hersteller freigegeben sein. /49/ Der Textinhalt sollte auch so gestaltet sein, dass dessen Sinn für den Betrachter ersichtlich ist, d.h. es sollten aussagekräftige Variablen- und Parameternamen enthalten sein.

5.2.2 Rechenbarkeit

Die Anforderung Rechenbarkeit ist für die Simulation am wichtigsten, da sie die korrekte Berechnung des Simulationsmodells in verschiedenen Simulationswerkzeugen fordert. Die Ausgabewerte eines bestimmten Signals dürfen nach der Transformation nicht von denen des ursprünglichen Modells abweichen. Dabei muss berücksichtigt werden, dass verschiedene Simulationswerkzeuge möglicherweise unterschiedliche numerische Lösungsverfahren anwenden. Einige Simulationswerkzeuge sind echtzeitfähig, so dass diese besondere Anforderungen an die Rechenbarkeit stellen.

Eine Übersicht über numerische Lösungsverfahren gibt Bild 5-6. Hier wird zunächst zwischen Ein- und Mehrschrittverfahren unterschieden. Während bei den Einschrittverfahren zur Lösung nur ein bekannter Wert erforderlich ist, wird bei den Mehrschrittverfahren noch auf weitere, zuvor berechnete Werte zurück gegriffen. Die Einschrittverfahren benötigen weniger Rechenzeit, so dass diese prinzipiell für Echtzeitanwendungen geeigneter sind. Andererseits weisen die mit den Mehrschrittverfahren berechneten Ergebnisse eine höhere Genauigkeit auf. /50/

Weiterhin wird zwischen expliziten und impliziten Lösungsverfahren unterschieden. Die Lösung der expliziten Verfahren ist nur vom aktuellen Wert abhängig, hingegen hängt diese bei den impliziten Verfahren bereits von dem gesuchten Wert ab, so dass derartige Probleme iterativ gelöst werden müssen. Analog zu den Einschrittverfahren sind die expliziten Rechenverfahren für Berechnungen in Echtzeit besser geeignet, während die impliziten Verfahren analog zu den Mehrschrittverfahren zeitaufwändiger sind, aber genauere Ergebnisse liefern. /50, 51/

	Einschrittverfahren	Mehrschrittverfahren
Explizit	Bsp. Euler: $x_{k+1} = x_k + h\Phi\,(t_k, x_k)$ k = 1,2	Bsp. Adams-Bashforth: $x_{k+1} = x_k + h\sum_{j=0}^{m} b_j\, f(t_{k-j}, x_{k-j})$ $b_j = \frac{(-1)^j}{j!\,(s-j)!}\int_0^1 \prod_{i=0, i\neq j}^{m} (u+i)\,du$ j=0, …, m
Implizit	Bsp. Euler: $x_{k+1} = x_k + h\Phi\,(t_{k+1}, x_{k+1})$ k = 0,1	Bsp. Adams-Moulton: $x_{k+1} = x_k + h\sum_{j=-1}^{m-1} b_j\, f(t_{k-j}, x_{k-j}), \quad 0 \le m \le n$ $b_j = \frac{(-1)^{j+1}}{(j+1)!\,(s-j-1)!}\int_0^1 \prod_{i=-1, i\neq j}^{m-1} (u+i)\,du$ j = -1, 0, …, m-1

Bild 5-6: Numerische Lösungsverfahren /50/

Die Anwendung verschiedener Lösungsverfahren kann somit zu Unterschieden bezüglich Stabilität, Genauigkeit, Echtzeitfähigkeit und Rechenaufwand führen. Es ist möglich, dass Modelle, die in einem Simulationswerkzeug stabil und mit hoher Genauigkeit laufen, jedoch unter geänderten Bedingungen (z.B. Echtzeitbedingungen) in einem anderen Simulationswerkzeug instabil werden, weil mit unterschiedlichen Lösungsverfahren gearbeitet wird.

Daher ist es wichtig sicherzustellen, dass das jeweilige Lösungsverfahren von allen eingesetzten Simulationswerkzeugen verwendet wird. Bei dem Fall, dass das Zielwerkzeug ein im Ausgangswerkzeug verwendetes numerisches Lösungsverfahren nicht unterstützt, wird dem Nutzer bei der Transformation ein entsprechender Hinweis gegeben. In diesem Fall muss ein anderes Lösungsverfahren im Ausgangswerkzeug gewählt werden, das im Zielwerkzeug vorhanden ist und mit dem sich das Modell stabil verhält, damit die Modelltransformation durchgeführt werden kann.

5.2.3 Wiedererkennbarkeit

Die Wiedererkennbarkeit von Modellen ist für die Modellnutzung in blockschaltbildbasierten Simulationswerkzeugen essenziell. Für den Anwender muss ein wiederverwende-

tes Modell auch in einem anderen als seinem gewohnten Simulationswerkzeug intuitiv wiedererkennbar sein. Die graphische Repräsentation eines Simulationsmodells im Blockschaltbild muss daher auch bei dessen Wiederverwendung in einem anderen Simulationswerkzeug beibehalten werden. Das bedeutet, die Modellblöcke müssen gleich aussehen bzw. intuitiv erkennbar sein, sie müssen gleich angeordnet sein und die Verbindungen der Blöcke untereinander müssen gleich sein. Die Unterteilung in Hierarchieebenen, z.B. in Form von Submodellen, muss ebenfalls erhalten bleiben. Durch die Wiedererkennbarkeit von Simulationsmodellen kann eine höhere Akzeptanz seitens der Anwender erreicht werden. Aus diesem Grund ist es bei der Modelltransformation essenziell, dass neben der reinen Modellinformation auch die graphische Darstellung transformiert wird.

Die graphische Repräsentation von Simulationsmodellen kann mit graphischen Benutzerschnittstellen verglichen werden. Die Anforderungen an eine grafische Benutzungsschnittstelle im Rahmen der Mensch-Computer-Kommunikation sind in der europäischen Norm EN ISO 9241-110 geregelt /52/. Dabei muss die Schnittstelle folgende Merkmale aufweisen /48/:

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Steuerbarkeit
- Erwartungskonformität
- Fehlertoleranz
- Individualisierbarkeit
- Lernförderlichkeit

Die Anforderung der Erwartungskonformität entspricht dabei genau der Anforderung, die im Rahmen dieser Arbeit als „Wiedererkennbarkeit“ bezeichnet wird. Sie ist wie folgt definiert:

„Ein Anwendungsprogramm ist erwartungskonform, wenn es den Kenntnissen aus bisherigen Arbeitsabläufen, der Ausbildung und Erfahrungen des Benutzers sowie allgemeinen Konventionen entspricht. Insbesondere heißt dies, dass die Anwendungsführung durchgehend konsistent gestaltet ist.“ /53/

Studien zur Untersuchung der Akzeptanz gegenüber der Einführung innovativer Prozesse in Unternehmen belegen, dass in der Benutzerakzeptanz zwar ein wesentlicher Erfolgsfaktor, zugleich jedoch die größte Schwierigkeit liegt. So ist davon auszugehen, dass die Gesamteffizienz eines Prozesses immer in direktem Zusammenhang mit der Benutzerakzeptanz steht. /54/

Simulationsmodelle, die nach der Wiederverwendung wiedererkennbar sind, sind für den Benutzer erwartungskonform. Somit erhöhen diese dessen Akzeptanz der Wiederverwendung.

5.2.4 Konfigurierbarkeit

Der Begriff „Konfiguration“ ist als eine „bestimmte Art der Gestaltung“ definiert, „Konfigurierbarkeit“ bezeichnet die Eigenschaft „sich konfigurieren zu lassen“ /55/. Unter „Konfigurierbarkeit“ im Zusammenhang von Simulationsmodellen ist somit die Möglichkeit zu verstehen, ein Simulationsmodell auf eine bestimmte Art und Weise zu verändern oder anzupassen.

Die Konfigurierbarkeit von Simulationsmodellen muss nach der Wiederverwendung erhalten bleiben. Der Nutzer muss die Möglichkeit haben, das wiederverwendete Modell zu editieren, das heißt er muss nicht nur Lesezugriff sondern auch Schreibzugriff auf das Modell haben. Zum einen soll das Einstellen globaler Parameter für das Gesamtmodell möglich sein. Globale Parameter enthalten Angaben zu Modellnamen, Integrationsverfahren, der eingestellten Rechenschrittweite etc. Zum anderen sollen lokale Parameter für einzelne Modellelemente durch den Nutzer veränderbar sein. Ebenso muss ein Hinzufügen oder Entfernen von Blöcken erfolgen können. Für den Nutzer soll hinsichtlich der Konfigurierbarkeit kein Unterschied zwischen einem Modell, das direkt in einem Simulationswerkzeug erstellt wurde und einem solchen, das in einem anderen Simulationswerkzeug wiederverwendet wird, festzustellen sein.

5.3 Zusammenfassung

In diesem Kapitel wurde das Konzept für die Wiederverwendung von Simulationsmodellen erarbeitet. Um nicht lediglich auf die Simulationsergebnisse eines Simulationsmodells zugreifen zu können, sondern eine vollwertige Wiederverwendung des Simulationsmodells erzielen zu können, soll eine Modelltransformation durchgeführt werden. Dazu wurden zwei mögliche Vorgehensweisen untersucht, die direkte Modelltransformation und die indirekte Modelltransformation über ein Metamodell. Beide Alternativen wurden hinsichtlich ihrer Effizienz, das heißt dem Verhältnis zwischen Nutzen und Aufwand, bewertet. Der Nutzen wurde dabei durch die Anzahl der Simulationswerkzeuge, in dem die Wiederverwendung erfolgen kann, der Aufwand mit der Anzahl der notwendigen Transformationen quantifiziert. Deren Verhältnis wurde als Kennzahl für die Effizienz der Wiederverwendung definiert. Die Transformation über ein Metamodell wurde als die effizientere Vorgehensweise identifiziert und als Vorgehensweise für das zu entwickelnde Konzept der Wiederverwendung festgelegt. Schließlich wurden Anforderungen an Simulationsmodelle herausgearbeitet, die zur Anwendung der gewählten Vorgehensweise erfüllt werden müssen.

6 Entwicklung des Meta-Metamodells

Auf Grund der in Kapitel 5 gewonnenen Erkenntnisse wurde für die Wiederverwendung von Simulationsmodellen die Vorgehensweise der Transformation über ein Metamodell festgelegt. Basierend auf den identifizierten Anforderungen an wiederverwendbare Modelle wird in diesem Kapitel ein geeignetes Metamodell entwickelt. Die große Herausforderung bei der Entwicklung eines Metamodells besteht darin, dass es zum einen so allgemeingültig sein muss, dass es für verschiedene Simulationswerkzeuge einsetzbar ist und zum anderen noch so speziell sein muss, dass keine spezifischen Informationen verloren gehen. Ein Modell, das noch eine Ebene allgemeiner beschrieben ist und somit die Struktur eines Metamodells beschreibt, wird als Meta-Metamodell bezeichnet /56/.

Im Rahmen dieser Arbeit werden die drei Simulationswerkzeuge Virtuos, Matlab/Simulink und SMP2, welche u.a. in den Domänen Maschinen- und Anlagenbau, Luft- und Raumfahrt eingesetzt werden und somit eine repräsentative Auswahl darstellen, in die WieMod-Workbench integriert. Diese werden in werkzeugspezifischen Metamodellen abstrahiert, bevor sie dann in einem übergeordneten Meta-Metamodell abgebildet werden. Die Entwicklung des Meta-Metamodells erfolgt nach dem Bottom-Up-Prinzip, bei dem ein Entwurf bei konkreten Objekten beginnt, die immer weiter abstrahiert werden. Wie diese Vorgehensweise zur Entwicklung des Meta-Metamodells angewendet wird, ist in Bild 6-1 von unten nach oben dargestellt.

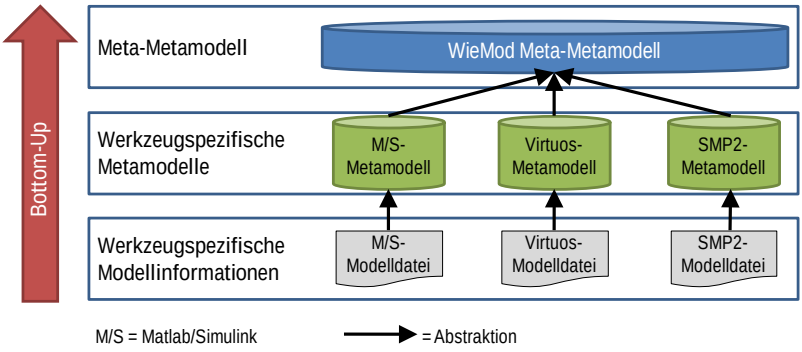


Bild 6-1: Entwicklung des WieMod Meta-Metamodells nach dem Bottom-Up-Prinzip

Um eine Modellbeschreibung entwickeln zu können, die Modelle der drei betrachteten Simulationswerkzeuge und einen Großteil weiterer werkzeugspezifischer Formate abbilden kann, wird zunächst die Modellstruktur der drei ausgewählten Simulationswerkzeuge analysiert. Auf Basis dieser Untersuchungen werden in drei werkzeugspezifischen Metamodellen allgemeine Modellbeschreibungen, die für den Großteil der Modelle des jeweiligen Simulationswerkzeugs gültig sind, hinterlegt. Schließlich kann ein übergeordnetes Meta-Metamodell, das WieMod Meta-Metamodell, entwickelt werden, das die werkzeugspezifischen Metamodelle abstrahiert. Somit kann es allgemeingültig die Modelle der drei betrachteten Simulationswerkzeuge sowie einen Großteil weiterer Modelle von hier nicht betrachteten Simulationswerkzeugen beschreiben.

Gemäß der Vorgehensweise nach dem Bottom-Up-Prinzip erfolgt in Kapitel 6.1 zunächst eine Analyse der Modelldateiformate der drei Simulationswerkzeuge. Die darauf basierend entwickelten, werkzeugspezifischen Metamodelle werden in Kapitel 6.2 erläutert. Kapitel 6.3 beschreibt den daraus abgeleiteten Aufbau des WieMod Meta-Metamodells detailliert.

6.1 Analyse werkzeugspezifischer Modellinformationen

In den folgenden drei Unterkapiteln werden die werkzeugspezifischen Modellinformationen der Simulationswerkzeuge Matlab/Simulink, Virtuos und SMP2 untersucht. Die Analyse erfolgt anhand des einfachen Beispielmodells eines Addierers, das in den drei Simulationswerkzeugen modelliert wurde.

6.1.1 Matlab/Simulink

Matlab ist ein Simulationswerkzeug, das von „The MathWorks, Inc.“ entwickelt wird. Es wird in verschiedenen Domänen wie dem Maschinen- und Anlagenbau, dem Fahrzeugbau, der Luft- und Raumfahrt etc. eingesetzt und überwiegend für technische Berechnungen, Datenanalysen und -auswertungen genutzt. Simulink gehört als eine Erweiterung von Matlab mit graphischer Bedienoberfläche zu den zahlreichen Toolboxen, mit denen Matlab für verschiedene Spezialanwendungen erweitert werden kann. In Simulink werden Simulationsmodelle graphisch als Blockschaltbild modelliert. /57/

Matlab/Simulink-Modelldateien werden in einem proprietären ASCII-Format mit der Dateiendung **.mdl* abgespeichert. Da Matlab/Simulink-Modelle blockschaltbildbasiert

sind, beinhaltet das Matlab/Simulink-Metamodell zusätzlich zu den semantischen Modellinformationen noch Informationen zur graphischen Repräsentation.

Die **.mdl*-Dateien beginnen mit einem Kopfteil (Header), der grundsätzliche Modellinformationen beinhaltet. Den grundsätzlichen Modellinformationen folgen konkrete Modellinformationen, die in semantische und geometrische Modellinformationen unterteilt sind. Der grundsätzliche Aufbau einer Matlab/Simulink-Modelldatei ist in Bild 6-2 dargestellt.

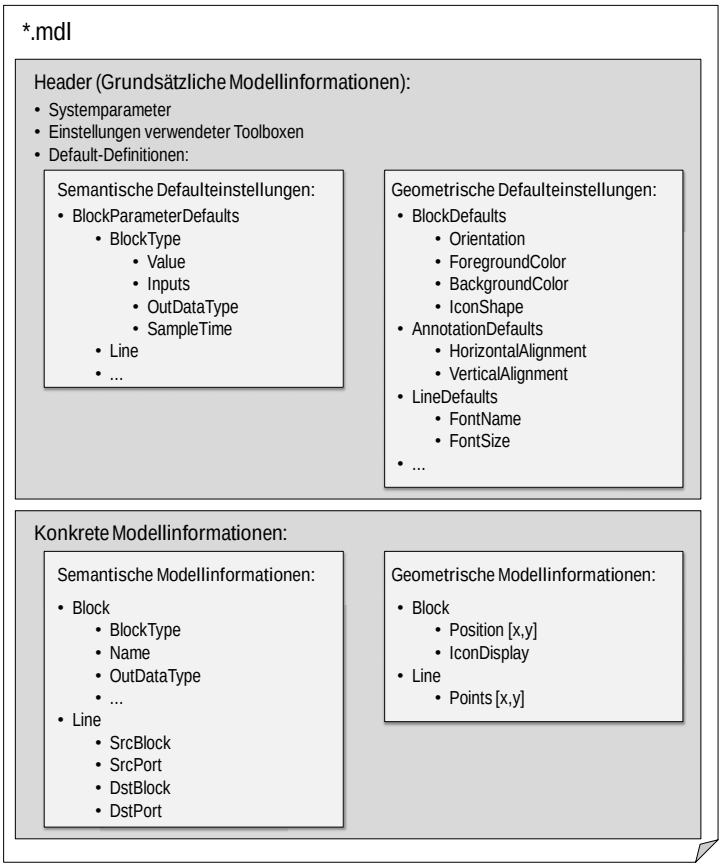


Bild 6-2: Aufbau einer Matlab/Simulink-Modelldatei

Im Header werden Systemparameter gesetzt, wie das verwendete numerische Lösungsverfahren, mögliche Testsignale oder der Name des Modellerzeugers. Außerdem finden sich in diesem Teil Einstellungen für die möglicherweise im Modell verwendeten Toolboxen. Weiterhin befinden sich im Header ausführliche Defaultdefinitionen für alle verwendeten Typen von Modellelementen. Da Matlab/Simulink blockschaltbasiert ist, handelt es sich bei den Modellelementen um Blöcke. Die Defaultdefinitionen der Modellelemente sind sehr ausführlich und enthalten Informationen zu allen möglicherweise enthaltenen Ein- und Ausgängen, Parametern, der möglichen Gestaltung der graphischen Repräsentation etc. Sie vervollständigen die später folgende, sehr knapp gehaltene Definition der Blöcke in den speziellen Modellen, bei der lediglich Angaben wie Blocktyp, dessen Name und geometrische Position im Blockschaltbild enthalten sind. Ebenso wie das gesamte Modell lassen sich die Defaultdefinitionen in semantische und geometrische Informationen einteilen. Die semantischen Defaulteinstellungen enthalten Informationen zu den Defaultparametern der im Modell enthaltenen Blöcke (*BlockParameterDefaults*). Werden bei der Modellierung vom Anwender keine anderen Parameterwerte angegeben, so werden die Defaulteinstellungen als Standardwerte übernommen. In der Defaultdefinition sind die Standardeinstellungen einzelner Blocktypen (*BlockType*) sowie der Signalverbindungen (*Line*) zwischen den Blöcken zu finden. In den geometrischen Defaulteinstellungen werden die Standardeinstellungen der graphischen Repräsentation beschrieben, wie zum Beispiel Form und Farbe der Blöcke sowie Schriftart und Schriftgröße der Beschriftungen im Modell.

Nach den grundsätzlichen Modellinformationen im Header folgen die konkreten Modellinformationen zu einem spezifischen Modell. Diese sind wie die Defaulteinstellungen in semantische und geometrische Modellinformationen unterteilt. Anders als bei den allgemeinen Defaulteinstellungen sind hier jedoch konkrete Informationen für ein spezifisches Modell hinterlegt. So werden in den semantischen Modellinformationen die in einem konkreten Modell enthaltenen Blöcke aufgelistet und durch Informationen zum Blocktyp, Namen und Datentypen ergänzt. Zudem wird spezifiziert, wie die einzelnen Blöcke miteinander verbunden sind. Der Parameter „SrcBlock“ (*Source Block*) legt fest, welcher Block als Quelle zu betrachten ist und von welchem Block somit eine Signalverbindung ausgehen kann. Der entsprechende Ausgangsport dieses Blocks wird mit „SrcPort“ (*Source Port*) bezeichnet. Entsprechend werden die Zielblöcke mit ihren Eingangsports „DstBlock“ (*Destination Block*) bzw. „DstPort“ (*Destination Port*) genannt. In den geometrischen Modellinformationen wird die konkrete graphische Repräsentation eines Modells festgelegt. Sofern keine Änderung durch den Nutzer vorgenommen wur-

de, werden die Defaulteinstellungen verwendet. Zusätzlich werden Angaben zu der genauen Position der einzelnen Blöcke und zum Verlauf der Signalverbindungen durch x-/y-Koordinaten festgelegt.

Anhand des Beispielmodells „AddConstants“ soll der Aufbau einer Matlab/Simulink-Modelldatei verdeutlicht werden. Das Modell besteht aus zwei Konstanten, die miteinander addiert werden. Das Ergebnis der Addition wird an dem Ausgang „Out1“ ausgegeben. (Bild 6-3)

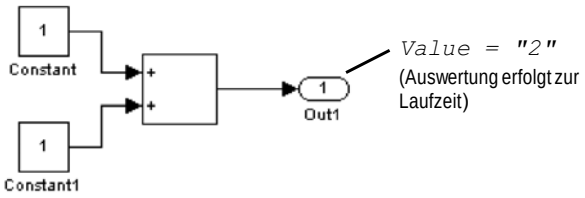


Bild 6-3: Beispielmodell "AddConstants" in Matlab/Simulink

Der folgende Codeausschnitt in Bild 6-4 zeigt, wie dieses Beispielmodell in einer Matlab/Simulink-Modelldatei abgespeichert wird.

```
System {
    Name                            "AddConstants"
    ...
    Block {
      BlockType                     Constant
      Name                          "Constant"
      Position                      [250, 120, 280, 150]
      Value                         "1"
      OutDataType                   "sfix(16)"
      OutScaling                    "2^0"
    }
    Block {
      BlockType                     Constant
      Name                          "Constant1"
      Position                      [245, 210, 275, 240]
      Value                         "1"
      OutDataType                   sfix(16)"
      OutScaling                    "2^0"
    }
    Block {
      BlockType                     Sum
      Name                          "Sum"
      Ports                         [2, 1]
      Position                      [315, 138, 375, 212]
      ShowName                      off
      Inputs                        "|++"
      InputSameDT                   off
      OutDataTypeMode               "Inherit via internal rule"
      OutDataType                   "sfix(16)"
      OutScaling                    "2^0"
      OutDataTypeStr                "Inherit: Inherit via internal rule"
      SaturateOnIntegerOverflow     off
    }
    Block {
      BlockType                     Outport
      Name                          "Out1"
      Position                      [425, 168, 455, 182]
      IconDisplay                   "Port number"
      OutDataType                   "sfix(16)"
      OutScaling                    "2^0"
    }
    Line {
      SrcBlock                      "Constant"
      SrcPort                       1
      Points                        [5, 0; 0, 40]
      DstBlock                      "Sum"
      DstPort                       1
    }
    Line {
      SrcBlock                      "Constant1"
      SrcPort                       1
      Points                        [20, 0]
      DstBlock                      "Sum"
      DstPort                       2
    }
    Line {
      SrcBlock                      "Sum"
      SrcPort                       1
      DstBlock                      "Out1"
      DstPort                       1
    }
  }
```

Bild 6-4: Codeausschnitt aus Matlab/Simulink-Modelldatei "*AddConstants.mdl*"

6.1.2 Virtuos

Virtuos ist ein Simulationswerkzeug, das von der Industriellen Steuerungstechnik GmbH entwickelt wird. Auf Grund seiner Echtzeitfähigkeit wird Virtuos bevorzugt für Hardware-in-the-Loop-Simulationen im Rahmen virtueller Inbetriebnahmen eingesetzt, bei der die reale Steuerung über einen realen Feldbus nicht an die reale Maschine, sondern an ein Simulationsmodell der Maschine gekoppelt wird.

Virtuos-Modelle werden in XML-basierten Modelldateien mit der Dateiendung **.ecf* beschrieben /58/. Virtuos-Modelle sind wie Matlab/Simulink-Modelle blockschaltbildbasiert, so dass neben semantischen Modellinformationen auch geometrische Informationen zur graphischen Repräsentation in einer Virtuos-Modelldatei enthalten sind. Tags in XML dienen zur Beschreibung und Deklaration der Modellelemente. Die XML-basierte Virtuos-Modelldatei hat das Root-Element <VirtuosMModel>. Die erste darauffolgende Ebene von Tags beinhaltet die sechs Elemente

- <DataType>
- <PeripherialSharedMemory>
- <Globals>
- <SimulationSettings>
- <ModelBlock> und
- <Geometry>.

Diese Elemente spezifizieren den Aufbau einer Virtuos-Modelldatei. Der Tag <DataType> beschreibt vordefinierte Grunddatentypen die nicht explizit im **.ecf*-File angegeben werden, wie zum Beispiel *VInt32*, *VReal32* oder *VVector3D*. Im Tag <PeripherialSharedMemory> erfolgt die Definition eines Named-Shared-Memory, der von peripheren IO-Ports zur Anbindung an einen Shared-Memory benutzt werden kann. In <Globals> werden globale Variablen definiert, die für die ganze Modelldatei gültig sind. Der Tag <SimulationSettings> beinhaltet generelle Simulationseinstellungen. Diese sind optional und überschreiben die Defaulteinstellungen des Modells. Die genannten vier Tags beinhalten grundsätzliche Modellinformationen. In <ModelBlock> wird ein Modellblock oder ein Submodell, das wiederum einen oder mehrere Modellblöcke enthalten kann, beschrieben. Jeder Tag <ModelBlock> beinhaltet seinerseits wiederum die Tags <Redefinitions> und <ModelStructure>. Wird ein Modellblock oder ein Submodell als Instanz eines Modellblocks aus der Blockbibliothek angelegt, dann können dessen Parameter,

Eigenschaften und die Namen von Ports im Abschnitt Redefinition abgeändert werden. Parameter, Eigenschaften und Portnamen die nicht redefiniert werden enthalten immer die Defaultwerte des Modellblocks aus der Bibliothek. In <ModelStructure> sind wiederum die Tags <Parameter>, <Property>, <Port> und <LinkageInput/Output> enthalten, die die Struktur des Blocks oder Submodells beschreiben. Die Gesamtheit dieser Tags definiert die Semantik eines Virtuos-Modells. Da Virtuos ein blockschaltbildbasiertes Simulationswerkzeug ist, wird die Geometrie des Blockschaltbilds zusätzlich im Tag <Geometry> beschrieben. (Bild 6-5)

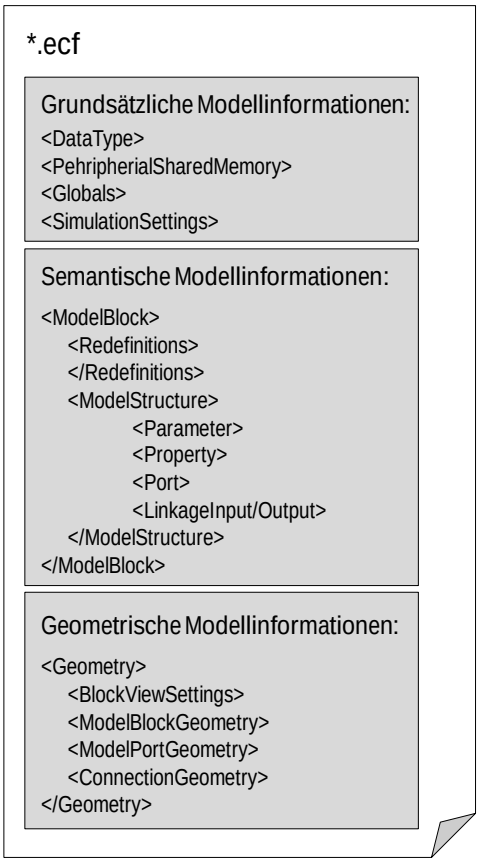


Bild 6-5: Aufbau einer Virtuos-Modelldatei

In Bild 6-6 ist das Beispielmodell mit dem Namen „AddConstants“ in Virtuos zu sehen. Das Modell besteht aus zwei Konstanten-Blöcken und einem Addierer-Block. Das Ergebnis der Addition wird an dem Ausgang „Out1“ ausgegeben.

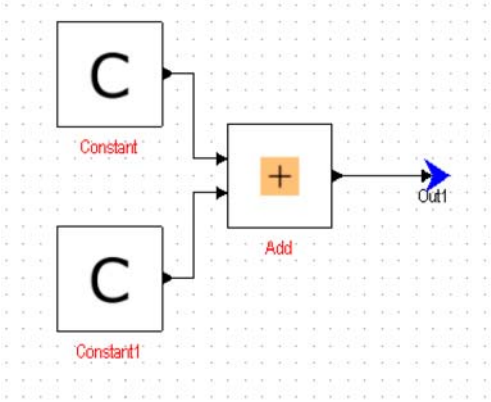


Bild 6-6: Beispielmodell "AddConstants" in Virtuos

Jedes enthaltene Modellelement wird in dem Tag <ModelBlock> mit Informationen zu Namen, UUID (Universal Unique Identifier), LinkageInputs und LinkageOutputs beschrieben. Die UUID ist eine eindeutige Identifizierungsnummer, mit der die einzelnen Elemente eindeutig unterschieden werden können. In der geometrischen Beschreibung des Blockschaltbilds wird die Geometrie der Elemente mit der jeweiligen UUID aus dem Semantikmodell referenziert.

Der folgende Codeausschnitt in Bild 6-7 zeigt, wie dieses Beispielmodell in einer Virtuos-Modelldatei abgespeichert wird. Es zeigt, wie die Modellbeschreibung, in der obiges Modell als Submodell mit Namen „AddConstants“ und dem Ausgangsport „Out1“ gespeichert ist. Zur besseren Übersichtlichkeit wurde die lange UUID jeweils durch „...“ ersetzt.

```
<?xml version="1.0" encoding="UTF-8"?>
<VirtuosMModel xmlns="http://de.wiemod.virtuos/Virtuos" version="1.4.1">
  <SimulationSettings/>
  <Globals/>
  <DataType/>
  <ModelBlock name="AddConstants" uuid="{…}">
    <ModelStructure>
      <Parameters/>
      <Properties/>
      <Ports>
        <Port ioType="output" name="Out1" type="VReal64" uuid="{…}"/>
      </Ports>
      <ModelBlock name="Constant" type="Constant" uuid="{…}">
        <ModelStructure>
          <Ports/>
        </ModelStructure>
        <Redefinitions>
          <Parameter name="Value" value="1"/>
        </Redefinitions>
        <LinkageInput name="Constant_out" portIoType="output" portName="out" uuid="{…}"/>
      </ModelBlock>
      <ModelBlock name="Constant1" type="Constant" uuid="{…}">
        <ModelStructure>
          <Ports/>
        </ModelStructure>
        <Redefinitions>
          <Parameter name="Value" value="1"/>
        </Redefinitions>
        <LinkageInput name="Constant1_out" portIoType="output" portName="out" uuid="{…}"/>
      </ModelBlock>
      <ModelBlock name="Sum" type="Add" uuid="{…}">
        <ModelStructure>
          <Ports/>
        </ModelStructure>
        <LinkageInput name="Sum_out" portIoType="output" portName="out" uuid="{…}"/>
        <LinkageOutput name="Constant_out" portIoType="input" portName="inp1" uuid="{…}"/>
        <LinkageOutput name="Constant1_out" portIoType="input" portName="inp2" uuid="{…}"/>
      </ModelBlock>
    </ModelStructure>
    <LinkageOutput name="Sum_out" portIoType="output" portName="Out1" uuid="{…}"/>
  </ModelBlock>
```

Bild 6-7: Codeausschnitt aus der Virtuos-Modelldatei "*AddConstants.ecf*"

6.1.3 SMP2

Die Modellbeschreibungssprache nach dem SMP2-Standard (Simulation Model Portability 2) wird in der Raumfahrt zur Beschreibung von Simulationsmodellen verwendet. Die Raumfahrt wird in dieser Arbeit exemplarisch als weitere Domäne neben dem Maschinen- und Anlagenbau betrachtet, um die domänenübergreifende Anwendbarkeit des erarbeiteten Konzepts zu demonstrieren. Für die Beschreibung eines Modells im SMP2-Standard sind mehrere XML-Dokumente notwendig. In einem übergreifenden XML-Dokument sind alle Dokumente, die zur Beschreibung eines Modells benötigt werden und ebenfalls auf XML basieren, aufgelistet. Ein *Catalogue* beinhaltet Typeninformatio-

nen des jeweiligen Modells. Im *Catalogue* „*Container.cat“* wird die Anzahl der enthaltenen Modellelemente definiert. Für jedes der enthaltenen Modellelemente existiert eine eigene *Catalogue*-Datei, in der das jeweilige Element mit seinem Namen sowie den verfügbaren Ein- und Ausgängen beschrieben ist. Die *Assembly* definiert, wie Modellelemente in einer spezifischen Konfiguration, die auf den Typeninformationen im *Catalogue* basiert, miteinander verbunden sind. Schließlich beinhaltet der *Schedule* Informationen zur Ausführungsreihenfolge der einzelnen Modellelemente.

Für das Beispielmodell „AddConstants“ werden im SMP2-Standard die in Bild 6-8 dargestellten XML-Dokumente benötigt. Da das Modell zwei verschiedenartige Modellelemente, nämlich einen Addierer sowie zwei Konstanten enthält, sind neben dem *Catalogue Container.cat* die *Catalogues Add.cat* und *Constant.cat* notwendig.

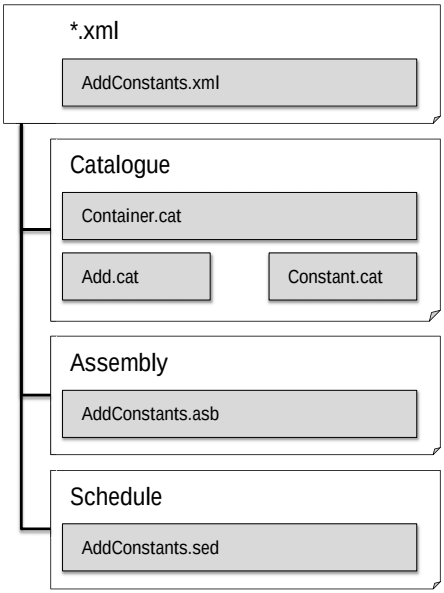


Bild 6-8: Benötigte Dokumente für das Beispielmodell "AddConstants"

In Bild 6-9 ist ein Codeausschnitt aus dem übergeordneten XML-Dokument *AddConstants.xml* zu sehen. In der Raumfahrt werden mehrere Simulationswerkzeuge eingesetzt, die Modelle im SMP2-Standard interpretieren können. In diesem Beispiel

wurde das Simulationswerkzeug „SimSat“ verwendet. Alle benötigten *Catalogue*-, *Assembly*- und *Schedule*-Dateien werden im Dokument *AddConstants.xml* aufgelistet.

```
<?xml version="1.0" encoding="UTF-8"?>
<Simsat>
...
        <Smp2Adapter>
          <Catalogues type="NoType" count="3">
            <Catalogue>AddConstants/Add.cat</Catalogue>
            <Catalogue>AddConstants/Constant.cat</Catalogue>
            <Catalogue>AddConstants/Container.cat</Catalogue>
          </Catalogues>
          <Assemblies type="NoType" count="1">
            <Assembly>AddConstants/AddConstants.asb</Assembly>
          </Assemblies>
          <Schedules type="NoType" count="1">
            <Schedule>AddConstants/AddConstants.sed</Schedule>
          </Schedules>
        </Smp2Adapter>
      </Component>
    </Models>
  </Kernel>
</Simsat>
```

Bild 6-9: Codeausschnitt aus der SimSat-Modelldatei "*AddConstants.xml*"

In Bild 6-10 ist ein Codeausschnitt aus dem *Assembly*-Dokument *AddConstants.asd* zu sehen. In diesem Dokument sind die drei Modellelemente mit ihren Parametern sowie die Verbindungen zwischen den einzelnen Modellelementen spezifiziert.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><Assembly:Assembly
Creator="WieMod SMP2 Transformator" Date="2012-02-11T17:33:33Z" Id="AddConstants"
Name="AddConstants" Version="1.0">
  <Model xlink:href="Container.cat#ID_..." xlink:role="Model"
   xlink:title="Model AddConstants"/>
  <Implementation>f0597721-4398-455e-8e7d-8a32cf93c7e4</Implementation>
  <ModelInstance Id="…" Name="Constant">
    <Model xlink:href="Constant.cat#Constant" xlink:role="Model"
     xlink:title="Model Constant"/>
          <Implementation>5ab36ce0-c657-43c7-b74a-0a1b136fbe82</Implementation>
    <FieldValue>
      <Field xlink:href="Constant.cat#Constant_Constant_P" xlink:title="Constant_P"/>
      <Value xsi:type="Types:StructureValue">
        <FieldValue>
          <Field xlink:href="Constant.cat#Constant_Parameters_Value"
           xlink:title="Value"/>
          <Value xsi:type="Types:SimpleValue">
            <Value xsi:type="xsd:double">1</Value>
          </Value>
        </FieldValue>
      </Value>    </FieldValue>
    <Container xlink:href="Container.cat#ID_..." xlink:role="Container"
     xlink:title="SystemContainer"/>
  </ModelInstance>
  <ModelInstance Id="…" Name="Constant1">
    <Model xlink:href="Constant.cat#Constant" xlink:role="Model"
     xlink:title="Model Constant1"/>
          <Implementation>5ab36ce0-c657-43c7-b74a-0a1b136fbe82</Implementation>
    <FieldValue>
      <Field xlink:href="Constant.cat#Constant_Constant_P" xlink:title="Constant_P"/>
      <Value xsi:type="Types:StructureValue">
        <FieldValue>
          <Field xlink:href="Constant.cat#Constant_Parameters_Value"
          xlink:title="Value"/>
          <Value xsi:type="Types:SimpleValue">
            <Value xsi:type="xsd:double">1</Value>
          </Value>    </FieldValue>
    <Container xlink:href="Container.cat#ID_..." xlink:role="Container"
     xlink:title="SystemContainer"/>
  </ModelInstance>
  <ModelInstance Id="ID_Instance_..." Name="Sum">
    <Model xlink:href="Add.cat#Add" xlink:role="Model" xlink:title="Model Sum"/>
    <Implementation>ec0c753f-fdee-4be8-b160-c6fc1abdc73c</Implementation>
    <Link Id="ID_FieldLink_..." Name="Connection" xsi:type="Assembly:FieldLink">
      <Input xlink:href="Add.cat#null" xlink:role="Input" xlink:title="1"/>
      <Source xlink:href="AddConstants.asb#ID_Instance_..." xlink:title="Constant"/>
      <Output xlink:href="Constant.cat#null" xlink:role="Output" xlink:title="1"/>
    </Link>
</ModelInstance>
</Assembly:Assembly>
```

Bild 6-10: Codeausschnitt aus der SimSat-Modelldatei "*AddConstants.asb*"

6.1.4 Vergleich der werkzeugspezifischen Modellinformationen

In diesem Unterkapitel werden die werkzeugspezifischen Modellinformationen der drei exemplarisch untersuchten Simulationswerkzeuge miteinander verglichen, um zu identifizieren, welchen Informationen unverzichtbar sind und zwingend in die werkzeugspezifischen Metamodelle und schließlich in das übergeordnete Meta-Metamodell übernommen werden müssen.

In Tabelle 6-1 sind die wesentlichen werkzeugspezifischen Modellinformationen zusammenfassend dargestellt.

	Matlab/Simulink	Virtuos	SMP2
Dateiformat	Proprietäres ASCII-Format	XML-basiert	XML-basiert
Modelldatei	*.mdl	*.ecf	*.xml, *.cat, *asb, *.sed
Semantische Modellinformationen	✓	✓	✓
• Modellstruktur	Unterteilung in Submodelle möglich	Unterteilung in Submodelle möglich	Unterteilung in Submodelle möglich
• Enthaltene Modellelemente	Typ Eingänge Ausgänge Parameter	Typ Eingänge Ausgänge Parameter	Typ Eingänge Ausgänge Parameter
• Verbindungen zwischen den Modellelementen	SrcBlock, DstBlock, SrcPort, DstPort	LinkageInput, LinkageOutput	xlink, im Dokument „Assembly (*.asb)“ definiert
Graphische Modellinformationen	✓	✓	✖
• Position	✓	✓	✖
• Ausrichtung	✓	✓	✖
• Farbe	✓	✓	✖
• Beschriftungen	✓	✓	✖

Tabelle 6-1: Vergleich der werkzeugspezifischen Modellinformationen

Matlab/Simulink-Modelldateien sind in einem proprietären ASCII-Format abgespeichert, Virtuos- und SMP2-Dateien sind hingegen XML-basiert. Während für Matlab/Simulink- und Virtuos-Modelle jeweils eine einzige Datei zur Speicherung eines

Modells ausreicht, werden für ein SMP2-Modell vier Dateien benötigt, auf die die unterschiedlichen Modellinformationen aufgeteilt werden.

Jedes der drei werkzeugspezifischen Modelle enthält semantische Modellinformationen, in denen Angaben zur Modellstruktur, den enthaltenen Modellelementen und der Verbindungen zwischen den Modellelementen abgespeichert sind. Bei allen drei Werkzeugen kann ein Modell in mehreren Submodellen strukturiert werden. Die jeweils enthaltenen Modellelemente werden über einen Typ, Ein- und Ausgänge sowie Parameter definiert. Die Definition der Verbindungen zwischen den Modellelementen erfolgt bei Matlab/Simulink und Virtuos in der jeweiligen Modelldatei, bei SMP2 ist diese Information in dem Dokument *Assembly* hinterlegt.

Der wesentliche Unterschied zwischen SMP2 und Virtuos und Matlab/Simulink besteht darin, dass Virtuos und Matlab/Simulink blockschaltbildbasiert sind und die Modelle somit eine graphische Repräsentation beinhalten. Deshalb enthalten diese Modelle neben den semantischen Modellinformationen zusätzliche geometrische Modellinformationen, mit denen Position, Ausrichtung und Farbe der Modellelemente sowie die Gestaltung deren Beschriftung definiert werden. SMP2-Modelle werden hingegen rein textuell und ohne graphische Repräsentation beschrieben, so dass neben semantischen Modellinformationen keine zusätzlichen geometrischen Modellinformationen vorhanden sind.

Die werkzeugspezifischen Metamodelle für Matlab/Simulink und Virtuos müssen deshalb, um alle werkzeugspezifischen Modelle beschreiben zu können, sowohl semantische als auch geometrische Modellinformationen enthalten. Das SMP2-Metamodell muss lediglich semantische Modellinformationen beinhalten, jedoch muss es in der Lage sein, die vier zur Definition eines SMP2-Modells benötigten Modelldateien zu beschreiben. Das übergreifende Meta-Metamodell muss sowohl semantische als auch geometrische Modellinformationen enthalten, damit es alle drei werkzeugspezifischen Metamodelle abbilden kann.

6.2 Entwicklung werkzeugspezifischer Metamodelle

Die werkzeugspezifischen Metamodelle für Matlab/Simulink, Virtuos und SMP2 sind in ihrer Struktur so aufgebaut wie die konkreten Modelldateien des jeweiligen Simulationswerkzeugs, jedoch in abstrahierter, allgemeinerer Form. Im Folgenden wird der Aufbau der drei umgesetzten werkzeugspezifischen Metamodelle erläutert. Zu Entwicklung der werkzeugspezifischen Metamodelle, muss eine formale Beschreibung der Modelle erfolgen, die für die werkzeugspezifischen Modelle zutreffend ist. Aus diesem Grund wurde in Kapitel 6.1 zunächst der Aufbau der werkzeugspezifischen Modelle anhand des einfachen Beispielmodells „AddConstants“ untersucht. Auf Basis dessen werden die wesentlichen Modellelemente identifiziert, die im Metamodell unbedingt abgebildet werden müssen, damit keine werkzeugspezifischen Informationen verloren gehen. Dennoch muss jedes werkzeugspezifische Metamodell noch so allgemein beschrieben sein, dass es durch das übergeordnete Meta-Metamodell abgebildet werden kann.

6.2.1 Matlab/Simulink-Metamodell

Das Matlab/Simulink-Metamodell ist eine Abstraktion der konkreten Matlab/Simulink-Modelle. Wie in der Analyse von Matlab/Simulink-Modellen in Kapitel 6.1.1 ersichtlich wurde, bestehen diese jeweils aus dem Header mit allgemeinen Definitionen der Modellelemente, die darauffolgend mit den konkreten semantischen und geometrischen Informationen des jeweiligen Modells ergänzt werden. Der Aufbau des Headers entspricht bereits dem Prinzip eines Metamodells, da es das Modell in abstrahierter Art und Weise beschreibt, ohne bereits konkrete Informationen des speziellen Modells zu enthalten. Somit kann dieser Aufbau als Grundlage für das Matlab/Simulink-Metamodell aufgegriffen werden, indem die allgemeinen Modellinformationen aus dem Header formal abgebildet werden. Matlab/Simulink-Modelle bestehen weiterhin aus semantischen Modellinformationen und einer zusätzlichen graphischen Repräsentation im Blockschaltbild. Aus diesem Grund muss auch das Matlab/Simulink-Metamodell sowohl semantische als auch geometrische Informationen abbilden können. Sehr vereinfacht betrachtet bestehen Matlab/Simulink-Modelle aus Blöcken, die über Signalverbindungen zwischen den Ein- und Ausgangsports der einzelnen Blöcke miteinander verbunden sind. Dies sind die wichtigsten Elemente eines konkreten Matlab/Simulink-Modells, die in dem zugehörigen Metamodell abgebildet werden müssen. Um die Eigenschaften eines Matlab/Simulink-Modells formal in einem Metamodell abbilden zu können, sind jedoch ne-

ben Blöcken (Block) und Signalverbindungen (Line) weitere Elemente notwendig, die in Bild 6-11 dargestellt sind.

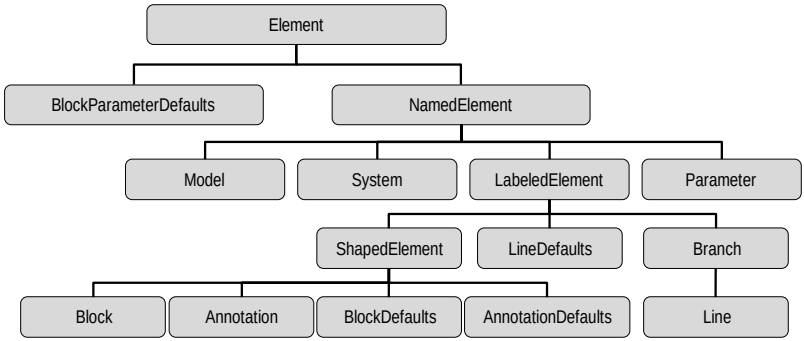


Bild 6-11: Elemente des Matlab/Simulink-Metamodells

Auf der linken Seite befindet sich das Element „BlockParameterDefaults“, in dem die Standardparameterwerte von Matlab/Simulink-Blöcken hinterlegt sind, die gelten, sofern keine abweichenden Angaben durch den Benutzer gemacht werden. Das Element „NamendElement“ enthält die Elemente „Model“, „System“, „LabeledElement“ und „Parameter“, die die konkreten Eigenschaften eines speziellen Modells beschreiben. Das Element „LabeledElement“ bezeichnet Elemente, die mit einer Beschriftung versehen werden können. Unterelemente eines „LabeledElements“ sind die Elemente „ShapedElements“, „LineDefaults“ sowie „Branch“. Das Element „Branch“ entspricht einer Signalverbindung, deren Standardeigenschaften in dem Element „LineDefaults“ beschrieben werden. Mit Hilfe des Elements „ShapedElement“ werden die geometrischen Eigenschaften der Elemente „Block“ und „Annotation“, was einer Notiz in einem Modell entspricht, definiert. Für Blöcke und Notizen sind wiederum Standarddefinitionen vorhanden, die in den Elementen „BlockDefaults“ und „AnnotationDefaults“ spezifiziert werden.

Das Klassendiagramm in Bild 6-12 stellt den Aufbau des Matlab/Simulink-Metamodells unter Verwendung der identifizierten unverzichtbaren Elemente eines Matlab/Simulink-Modells als Klassen mit dazugehörigen Attributen in detaillierter Form dar.

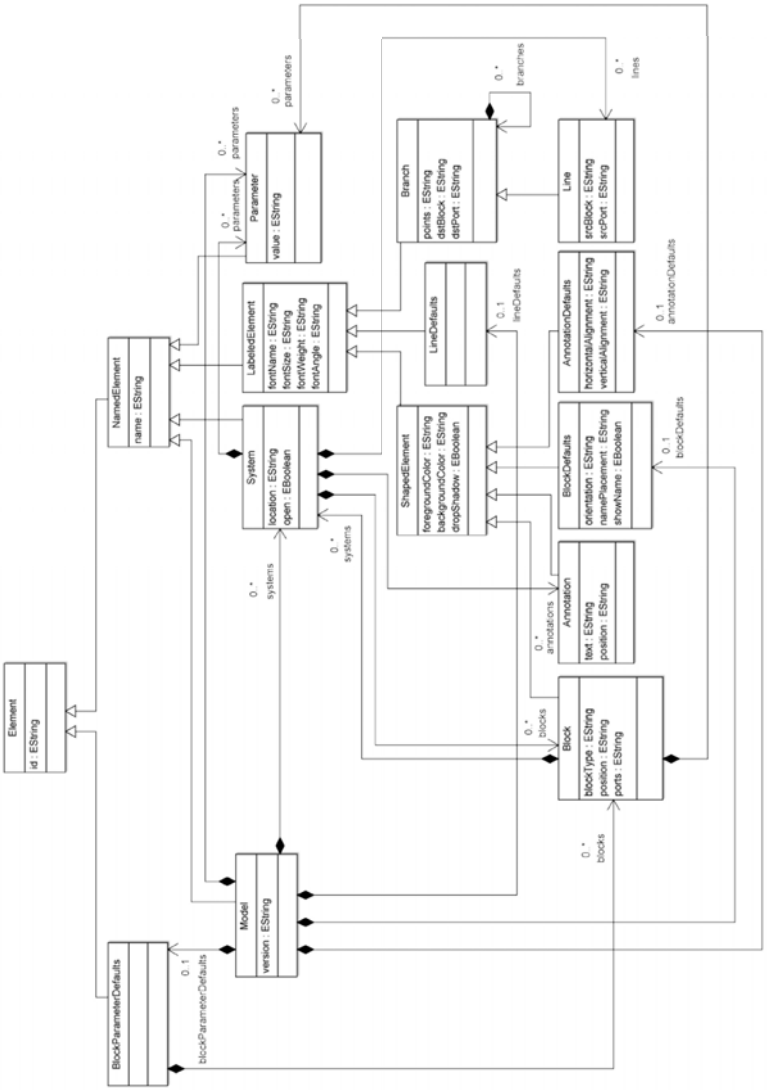


Bild 6-12: Aufbau des Matlab/Simulink-Metamodells

6.2.2 Virtuos-Metamodell

Das Virtuos-Metamodell ist eine Abstraktion der konkreten Virtuos-Modelle. Da Virtuos-Modelle blockschaltbildbasiert sind, beinhaltet das Virtuos-Metamodell genau wie Virtuos-Modelldateien zusätzlich zu den semantischen Modellinformationen noch geometrische Modellinformationen zur graphischen Repräsentation des Blockschaltbilds.

Das Virtuos-Metamodell ist strukturell genauso aufgebaut wie das Matlab/Simulink-Metamodell und besteht aus den gleichen Elementen, so dass die Darstellung aus Bild 6-11 für das Virtuos-Metamodell ebenfalls zutreffend ist.

Das Klassendiagramm in Bild 6-13 veranschaulicht den Zusammenhang der Modellelemente des Virtuos-Metamodells. Da die Modellelemente der Metamodelle für Matlab/Simulink und Virtuos exakt übereinstimmen, ist auch der Aufbau der Klassendiagramme prinzipiell identisch. Die werkzeugspezifischen Unterschiede sind in den Attributen der Klassen „Element“, „System“, „Parameter“, „Branch“ und „Block“ hinterlegt.

Aufgrund der großen Gleichteile des Matlab/Simulink- und des Virtuos-Metamodells ergeben sich auch für die Entwicklung des Meta-Metamodells Synergien. Die Gleichteile können als Schnittmenge der Modellbeschreibung direkt in das Meta-Metamodell übernommen werden und somit zur Beschreibung werkzeugspezifischer Modelle für sowohl Matlab/Simulink als auch Virtuos verwendet werden. Die werkzeugspezifischen Unterschiede, die sich in den geringfügigen Unterschieden der beiden Metamodelle zeigen, müssen in ihrer Gesamtheit im Meta-Metamodell berücksichtigt werden, so dass die werkzeugspezifischen Eigenschaften der Modelle beider Simulationswerkzeuge mit diesem beschrieben werden können.

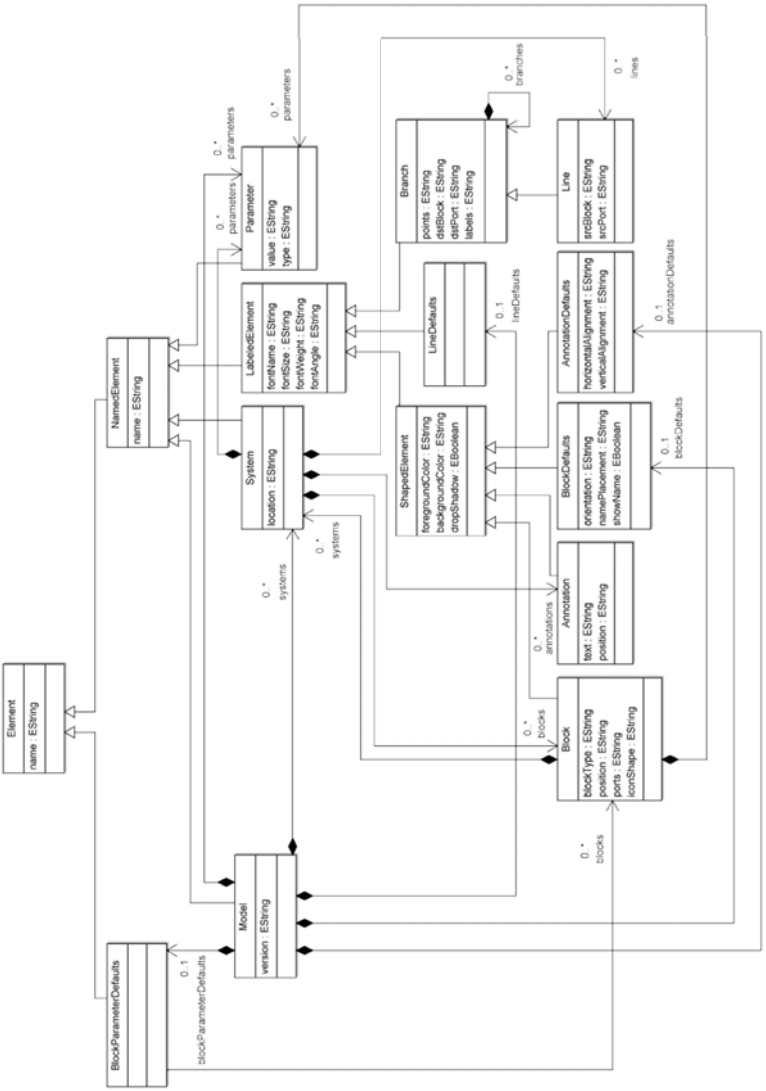


Bild 6-13: Aufbau des Virtuos-Metamodells

6.2.3 SMP2-Metamodell

Das SMP2-Metamodell ist als Abstraktion der konkreten SMP2-Modelle ebenfalls XML-basiert. Da SMP2-Modelle anders als Matlab/Simulink- und Virtuos-Modelle rein textuell beschrieben sind und keine graphische Repräsentation enthalten, besteht das SMP2-Metamodell entsprechend nur aus einem Semantikmodell und enthält kein zusätzliches Layoutmodell.

Zum SMP2-Standard existiert bereits ein Metamodell, das von der ESA (European Space Agency) definiert wurde. Das SMP2-Metamodell stellt fünf Arten von SMDL-Dokumenten (Simulation Model Definition Language) zur Verfügung, die jeweils einen bestimmten Zweck erfüllen: *Catalogues* zur Definition von Modelltypen, *Assemblies* zur Integration von Modellinstanzen, *Schedules* zur Terminierung und Ausführung von Assemblies, *Packages* zur Definition der Kapselung von Implementierungen und *Workspaces* zur Definition der Gruppierung von SMDL Dokumenten. /59/

Diese Dokumente sind voneinander abhängig: Die Definition von Modelltypen in einem *Catalogue* legen fest, wie die Modellinstanzen später in einer *Assembly* miteinander verknüpft werden können. *Namespaces* bilden in einem *Catalogue* den primären Ordnungsmechanismus. Diese können verschachtelt sein und so eine Hierarchie bilden. Die Terminierung der Modelle wird durch die in einer *Assembly* verfügbaren Modellinstanzen und deren Typendefinition in den zugehörigen *Catalogues* eingegrenzt. Jedes implementierte Element in einem *Package* referenziert einen Typ in einem Catalogue. Ein *Workspace* beinhaltet Verweise zu jeder anderen der vier Dokumente. Alle Dokumenttypen verwenden einfache Elemente, die getrennt in einem dedizierten XML Schema (*Core Elements*) gespeichert sind und von allen anderen Schemadokumenten eingebunden werden. /59/

Die Simulation Model Definition Language (SMDL) ist die Beschreibungssprache für SMP2-konforme Typen und Modelle, sowie *Assemblies* und *Schedules*. Während das SMP2-Komponentenmodell die Interaktion von SMP2-Modellen und anderen SMP2-Komponenten definiert, bildet diese Sprache die Basis für die Zusammenarbeit von SMP2-konformen Tools. Sie ist in UML spezifiziert und in einer Reihe von XML-Schemen, die automatisch aus dem UML-Modell generiert werden, implementiert. Aus diesem Grund können SMDL-Modellbeschreibungen in XML-Dokumenten syntaktisch gegen die generierten Schemen validiert werden. Diese Vorgehensweise stellt sicher,

dass SMDL-Modellinformationen sicher zwischen verschiedenen Interessengruppen ausgetauscht werden können. /59/

In Bild 6-14 ist der Aufbau des SMP2-Metamodells dargestellt.

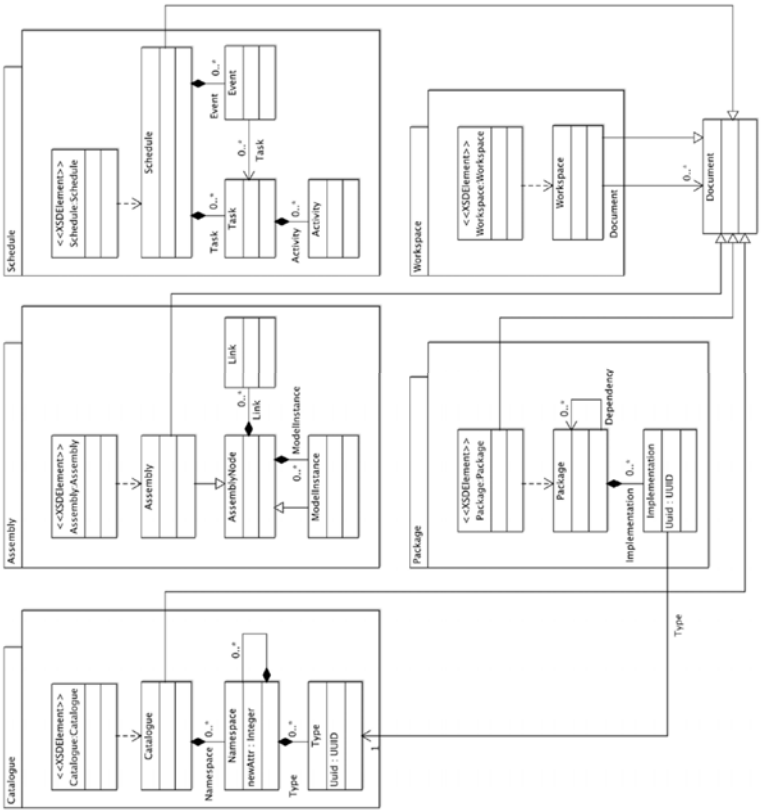


Bild 6-14: Aufbau des SMP2-Metamodells in Anlehnung an /59/

Der Aufbau des SMP2-Metamodells weicht von dem der Metamodelle für Matlab/Simulink und Virtuos ab (vgl. Bild 6-12 und Bild 6-13). Um die Eigenschaften eines SMP2-Modells formal in einem Metamodell so abbilden zu können, dass es in das über-

greifende WieMod Meta-Metamodell integriert werden kann, werden die in Bild 6-15 dargestellten Elemente verwendet.

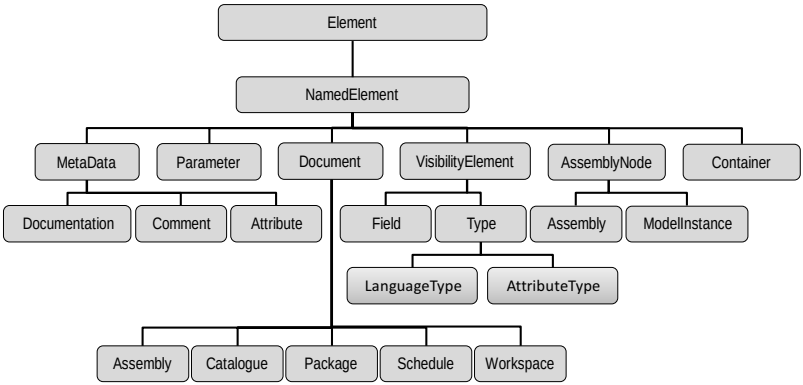


Bild 6-15: Elemente des SMP2-Metamodells

Das Element „NamendElement“ enthält die Elemente „MetaData“, „Parameter“, „Document“, „VisibilityElement“ und „AssemblyNode“. Das Element „MetaData” enthält Zusatzinformationen, um ein „NamedElement“ mit Anmerkungen zu versehen. Dazu enthält es die Unterelemente „Documentation“, „Comment“ und „Attribute“. Das Element „Documentation“ stellt zusätzliche Dokumente über Links zu technischen Zeichnungen, 3D-Animationen etc. zur Verfügung. Das Element „Comment“ enthält Nutzerkommentare, zum Beispiel zum Review eines Modells. Mit Hilfe des Elements „Attribute“ können weitere nutzerdefinierte Metadaten zu einem „NamedElement“ hinzugefügt werden.

Das Element „Document“ ergänzt Dokumente um Informationen zu Titel, Datum, Autor und Versionsnummer und enthält die zuvor beschriebenen Unterelemente „Assembly“, „Catalogue“, „Package“, „Schedule“ und „Workspace“.

Ein “VisibilityElement” ist Element, dessen Sichtbarkeit eingeschränkt werden kann. Die Sichtbarkeit kann global (*public*), lokal auf das Elternelement beschränkt (*private*), lokal auf das Elternelement und davon abgeleitete Typen beschränkt (*protected*) oder innerhalb eines Packages (*package*) beschränkt sein.

Das Element “AssemblyNode” ist der zentrale Punkt der Semantik eines SMP2-Modells und enthält die Elemente „Assembly“ und „ModelInstance“. Das Element “Assembly” beschreibt ein Modell, das von einem bestimmten Catalogue spezifiziert wird. Das Element “ModelInstance” repräsentiert eine Instanz eines Modells. Aus diesem Grund muss es einen Link zu dem spezifischen Modell enthalten. Jede Instanz eines Modells befindet sich entweder in einer *Assembly* oder in einer anderen Instanz eines Modells.

Ein „Container“ definiert Regeln für die Zusammensetzung eines Modells mit der Anzahl der enthaltenen Elemente.

6.3 Entwicklung des WieMod Meta-Metamodells

Das Meta-Metamodell abstrahiert die werkzeugspezifischen Metamodelle. Die werkzeugspezifischen Metamodelle für Matlab/Simulink und Virtuos enthalten sowohl semantische als auch geometrische Modellinformationen, da die werkzeugspezifischen Modelle blockschaltbildbasiert sind. Im Gegensatz dazu sind Modelle nach dem SMP2-Standard rein textuell beschrieben, so dass im entsprechenden Metamodell lediglich semantische Modellinformationen enthalten sind. Um diese beiden Arten von Metamodellen in einem gemeinsamen Meta-Metamodell beschreiben zu können, muss dieses sowohl ein Semantik-Modell (Kapitel 6.3.1) als auch ein zusätzliches Layout-Modell (Kapitel 6.3.2), in dem geometrische Informationen zum Layout des Blockschaltbilds hinterlegt werden, beinhalten. Wie die werkzeugspezifischen Modelle und Metamodelle wird das Meta-Metamodell ebenfalls in einem textuellen Format gespeichert, so dass auch hier die Anforderung nach Lesbarkeit erfüllt werden kann. Das Meta-Metamodell kann sowohl für einzelne Simulationsmodellelemente verwendet werden, als auch komplette Modelle darstellen. Komplette Modelle können weitere Parameter besitzen, wie Informationen über das verwendete numerische Lösungsverfahren oder die Lizenz des Modells.

Das entwickelte Meta-Metamodell, das die werkzeugspezifischen Metamodelle für Matlab/Simulink, Virtuos und SMP2 und damit auch die werkzeugspezifischen Modelle für diese drei Simulationswerkzeuge beschreiben kann, ist in Bild 6-16 dargestellt. Dessen Unterteilung in ein Semantik- und ein Layout-Modell wird in den folgenden Unterkapiteln detailliert beschrieben.

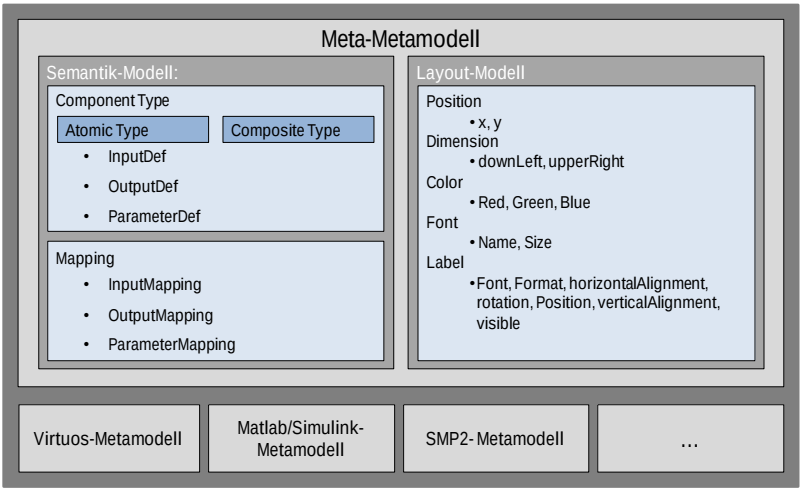


Bild 6-16: Aufbau des Meta-Metamodells /12/

6.3.1 Semantik-Modell

Das Semantik-Modell enthält alle logischen Modellinformationen. Die Inhalte eines Simulationsmodells sind hier abgespeichert. Das Semantik-Modell besteht aus zwei Teilen. Im ersten Teil werden die enthaltenen Komponenten mit ihren Eingängen, Ausgängen und Parametern definiert. Im zweiten Teil werden die Beziehungen zwischen den Komponenten unterschiedlicher Simulationswerkzeuge in den so genannten Mappings hergestellt. Mit diesem Teil des Meta-Metamodells werden die in den einzelnen Simulationsmodellen verwendeten Eingänge, Ausgänge und Parameter beschrieben und durch die Mappings in anderen Simulationswerkzeugen wiederverwendbar. Das Semantik-Modell ermöglicht somit die Erfüllung der Anforderungen nach Rechenbarkeit und Konfigurierbarkeit. Das entwickelte Semantik-Modell ist in Bild 6-17 in einem Klassendiagramm zusammengefasst. Die einzelnen Klassen sowie deren Beziehungen untereinander werden in diesem Unterkapitel beschrieben.

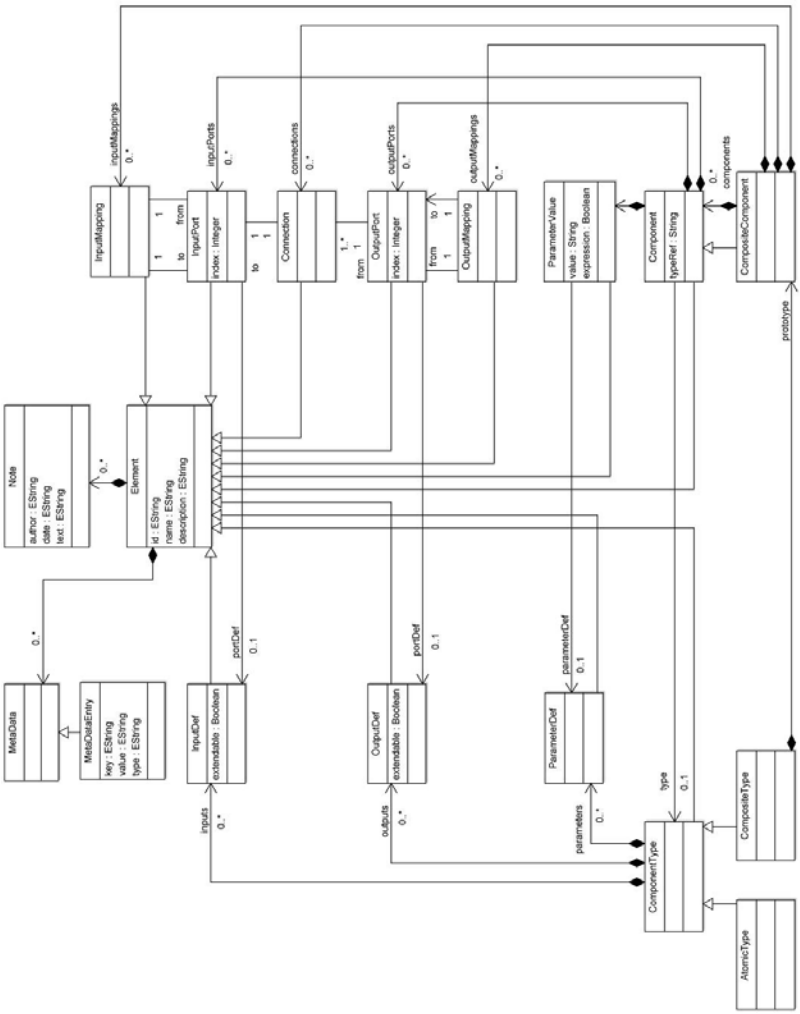


Bild 6-17: Semantik-Modell

Auf der linken Seite des Klassendiagramms befinden sich die Klassen, die zum Typ einer Komponente, d.h. eines Modellelements, gehören. Diese sind für alle Instanzen eines Typs gleich und beschreiben in abstrakter Form die Schnittstelle, die eine Komponente besitzt. Auf der rechten Seite erweitern die Klassen den Typen um Informationen zur konkreten Instanz, wie die Parameterbelegung oder Verbindungen zwischen Komponenten.

Alle Klassen des Semantik-Modells erben von der Klasse Element einen Namen, eine Beschreibung und die Möglichkeit, auf Metadaten (Key-Value-Paare) und Notizen (Text mit Autor und Datum) von Komponenten zuzugreifen. ParameterValue referenziert eine Parameterdefinition, die für alle Instanzen dieses Bausteintyps den Namen vorgibt.

Alle Komponenten stellen ihre Funktionalität über Eingangs- und Ausgangsports zur Verfügung. Die Typdefinitionen dienen dazu, die Schnittstelle und die möglichen Parameter der Komponententypen zu beschreiben. Die Funktionalität der einzelnen Klassen wird im Folgenden beschrieben.

Die Klasse *ComponentType* beinhaltet die zentrale Beschreibung eines Komponententyps. Dabei kann es sich entweder um *AtomicTypes* oder *CompositeTypes* handeln. *AtomicTypes* sind Typen, die nicht weiter unterteilbar sind. Sie repräsentieren den kleinsten Baustein des Systems. Im Gegensatz dazu sind *CompositeTypes* Typen, die aus anderen Komponenten zusammengesetzt sind. Sie enthalten innere Komponenten, so genannte *Subcomponents*, und Verbindungen.

In den Klassen *InputDef* und *OutputDef* werden Signal-Ein- und Ausgänge für Typen definiert. Falls die Anzahl der Ein- und Ausgangs-Ports nicht fix ist, d.h. das Attribut *extendable* gesetzt ist, und somit veränderbar ist, können in der konkreten Komponente mehrere Ports angegeben werden. Die Einheiten der Ports werden als Metainformation gespeichert. Die Klasse *ParameterDef* definiert Parameter für einen Typ und beinhaltet einen Defaultwert.

Die folgenden Klassen beinhalten jeweils zusätzliche Informationen zur Instanz-Ebene der Komponententypen. Jeder Baustein wird durch diese Klassen dargestellt und durch die Parameter bestimmt.

Die Klasse *Component* repräsentiert ein ganzes Modell oder eine Modellkomponente. Eine *Component* hat Input- und Output-Ports und Parameterbelegungen. Eine *CompositeComponent* ist die Oberkomponente für komplexere Modelle und gleichzeitig eine *Component* für größere Modelle. Sie erbt die Eigenschaften der Klasse *Component* und besteht zusätzlich aus *Subcomponents* und den Verknüpfungen der Output- und Input-Ports. Die Klasse *ParameterValue* definiert den Wert eines Parameters. Jeder konkrete Parameter kann sowohl einen mathematischen Ausdruck, wenn das Attribut *Expression* gesetzt ist, oder einen festen Wert annehmen.

Die Klassen *OutputPort* und *InputPort* beschreiben die Eingangs- und Ausgangsports einer Komponente. Die Ports repräsentieren die Verbindungsmöglichkeiten der Komponenten und dienen als Startpunkt aller vorhandenen Verbindungen. Falls die Portdefinition vorsieht, dass mehrere Instanzen eines Porttyps angelegt werden dürfen, d.h. das Attribut *extendable* gesetzt ist, so können die einzelnen Ports über ihren Index eindeutig bestimmt werden.

Die drei Verknüpfungen *Connection*, *InputMapping* und *OutputMapping* sind jeweils direkte Verbindungen zwischen Ports und unterscheiden sich in den Port-Typen, die als Quelle (*from*) und als Ziel (*to*) zulässig sind. Parameter können immer von einer Subkomponente über Parameter der übergeordneten Komponente abgebildet werden. Dies geschieht entweder direkt über Angabe des Parameternamens der Eltern-Komponente oder, bei komplexen Parametern, über Funktionen, die auch mehrere Parameter und Umformungen beinhalten können.

An allen Elementen des semantischen Modells (außer *ParameterValue*) können generische Metadaten gespeichert werden. Metadaten bestehen aus Key-Value-Paaren und haben einen Typ (*String*, *Integer*, *Boolean*). Metadaten können in Metadaten-Kategorien gruppiert werden.

Mit dem entwickelten Semantik-Modell kann nun die Semantik der werkzeugspezifischen Metamodelle und damit die Semantik beliebiger Modelle aus den drei betrachteten Simulationswerkzeugen beschrieben werden.

6.3.2 Layout-Modell

Mit diesem Teil des Meta-Metamodells wird die Erfüllung der Anforderungen nach Wiedererkennbarkeit erreicht. Hier werden Zusatzinformationen, wie der Verlauf der Verbindungen und die Anordnung der Komponenten in einer zweidimensionalen Darstellung abgespeichert. Solche graphischen Eigenschaften der Komponenten sind im Semantik-Modell nicht enthalten, sondern werden separat im Layout-Modell hinterlegt. Das Layout-Modell kann bei Typdefinitionen mit in das Modell aufgenommen werden und wird, falls vorhanden, bei allen anderen Komponenten referenziert. Die Klassen können in Hilfsklassen und Modellklassen unterteilt werden. Die **Hilfsklassen** stellen grundsätzliche Layout-Informationen, wie 2D-Positionen, Farben oder Textausrichtungen dar. Ihre Bedeutung ergibt sich aus dem Kontext, in dem sie verwendet werden. Die Hilfsklassen sind in der folgenden Tabelle zusammengefasst.

Klasse	**Parameter**	**Beschreibung**
Position	x, y	Ganze Zahlen für eine 2D-Position
Dimension	downLeft, upperRight	Die Dimension eines Rechtecks
Color	Red, Green, Blue	Farben werden als RGB-Werte [0,255] gespeichert
Font	Name, Size	Name der Schriftart und Größe der Schrift
Label	Font, Format, horizontalAlignment, rotation, Position, verticalAlignment, visible	Beschreibung der Texteigenschaften wie Ausrichtung, Schriftart oder Format

Tabelle 6-2: Hilfsklassen des Layout-Modells

In den **Modellklassen** werden mit Ausnahme der Klasse *ParameterValue* alle Klassen aus dem Semantikmodell um Layoutinformationen erweitert. Äquivalent zu den Metadaten im semantischen Modell, können auch Layoutinformationen mit Metadaten erweitert werden. Dies ist zum Beispiel bei blockspezifischen Informationen der Fall, die nicht explizit im Meta-Metamodell vorgesehen sind.

Komponenten werden als ein Rechteck über die Position der Eckpunkte beschrieben. Die unterschiedlichen Formen liegen komplett innerhalb dieses Rechtecks. Zusätzlich

sind eine Orientierung für die Ausrichtung der Schrift, eine Hinter- und Vordergrundfarbe und eine Formatbeschreibung für den Text definiert. Der Name der Formatbeschreibung wird ignoriert. Zusammengesetze Komponenten (*CompositeComponents*) haben zusätzlich eine innere Dimension für das Subsystem und können Notizen (*Annotations*) enthalten.

Input- und Output-Ports werden durch ihre absolute Position beschrieben. Ihr Name wird nur angezeigt, wenn das Attribut *nameVisible* gesetzt ist. Ebenso wie die Input- und Output-Mappings werden die zugehörige Komponente im semantischen Modell und der Port referenziert. Bei In- und Output-Mappings kann zusätzlich eine Block-Position angegeben werden, die einen Block innerhalb der umgebenden *CompositeComponent* angibt. Verbindungen setzen sich aus einem Startpunkt, beliebig vielen Zwischenpunkten und Verzweigungen sowie Endpunkten zusammen. Der Startpunkt kann entweder ein *InputMapping* oder ein *OutputPort* sein. Die Zwischenschritte (*ConnectionStep*) können beliebig viele Stützpunkte (*position*) und für jeden Abschnitt zwischen den Stützpunkten eine Labelposition besitzen. Auf einen Zwischenschritt können weitere Zwischenschritte folgen, was einer Verzweigung entspricht, oder ein Verbindungsende. Die Verbindungsenden sind *OutputMappings* oder *InputPorts*.

Als letzte Klasse von Layout-Komponenten gibt es Notizen, die keine Entsprechung im semantischen Modell haben. Sie bestehen aus einer Position, einem Namen und einem Text inklusive Formatierung, Vorder- und Hintergrundfarbe und Ausrichtung.

Mit dem entwickelten Layout-Modell kann die graphische Repräsentation beliebiger Simulationsmodelle der Werkzeuge Matlab/Simulink und Virtuos beschrieben werden. Für die Beschreibung von SMP2-Modellen wird das Layout-Modell nicht benötigt, da diese Modelle rein textuell beschrieben sind die Beschreibung mit dem Semantik-Modell ausreicht.

Somit wurde ein Meta-Metamodell entwickelt, das mit der Kombination aus Semantik- und Layout-Modell alle drei werkzeugspezifischen Metamodelle und somit die werkzeugspezifischen Modelle abbilden kann. Die Anforderungen nach Lesbarkeit, Rechenbarkeit, Wiedererkennbarkeit und Konfigurierbarkeit können mit diesem Meta-Metamodell erfüllt werden. Dies wird im Rahmen der Validierung in Kapitel 8.2 an einem Beispielmodell demonstriert.

7 Methoden zur Wiederverwendung von Simulationsmodellen

In diesem Kapitel werden konkrete Methoden zur Wiederverwendung von Simulationsmodellen nach dem in Kapitel 5 ausgewählten Konzept der Wiederverwendung über das in Kapitel 6 entwickelte Meta-Metamodell erarbeitet. Auf Grund der indirekten Transformation der Modelle über das Meta-Metamodell erfolgt die Wiederverwendung in zwei Schritten, die in Bild 7-1 dargestellt sind.

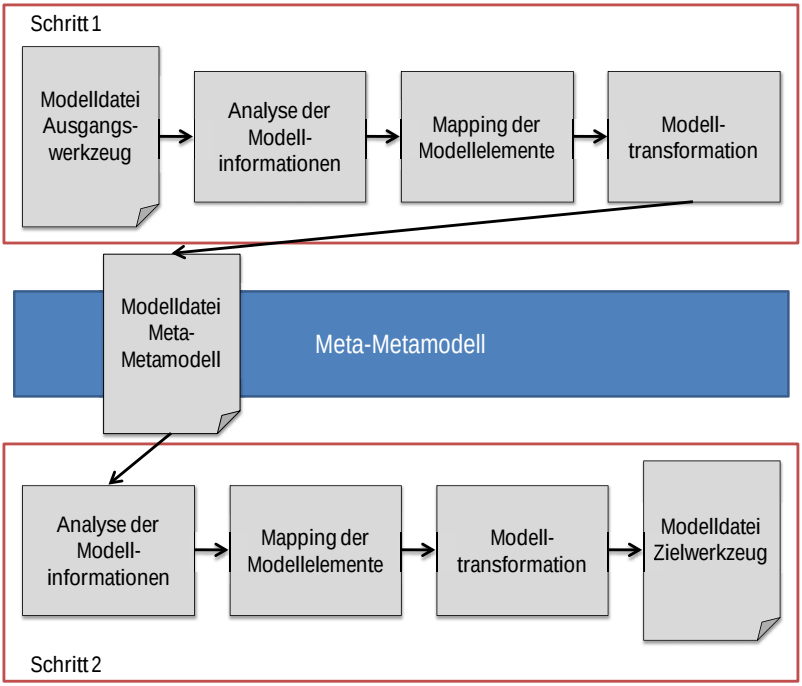


Bild 7-1: Wiederverwendung von Simulationsmodellen in zwei Schritten über das Meta-Metamodell

Im ersten Schritt erfolgt eine Modelltransformation vom Ausgangswerkzeug zum Meta-Metamodell. Dabei wird die Modelldatei eines Ausgangswerkzeugs hinsichtlich der darin enthaltenen Modellinformationen, wie sie in Kapitel 6.1 beispielhaft für die Simulationswerkzeuge Virtuos, Matlab/Simulink und SMP2 beschrieben wurden, analysiert.

Anschließend erfolgt ein Mapping der Modellelemente, jedoch nicht direkt auf die entsprechenden Modellelemente des Zielwerkzeugs, sondern zunächst auf die entsprechenden Modellelemente des Meta-Metamodells. Dazu werden Strategien zum Mapping von Modellelementen erarbeitet. Auf Basis des Mappings kann die eigentliche Transformation der Simulationsmodelle von einem Simulationswerkzeug in ein anderes erfolgen. Diese erfolgt analog zu dem Mapping jedoch ebenfalls nicht direkt in eine Modelldatei für das gewünschte Zielwerkzeug, sondern zunächst in eine Modelldatei des Meta-Metamodells.

Im zweiten Schritt wird die gesamte Prozesskette des ersten Schritts erneut durchlaufen. Die Modellinformationen der Modelldatei im Meta-Metamodell, die als Ergebnis des ersten Schritts generiert wurde, werden nun analysiert. Im Anschluss erfolgt wieder ein Mapping der Modellelemente mit dem Unterschied, dass das Mapping diesmal von den Modellelementen des Meta-Metamodells auf die entsprechenden Modellelemente des Zielwerkzeugs erfolgt. Dieses Mapping wird im finalen Schritt als Modelltransformation zu einer Modelldatei für das Zielwerkzeug angewandt.

Im Folgenden werden zusätzlich zu den Ausführungen in Kapitel 2.1.3 weitere Begriffe definiert, die zur Beschreibung der Mapping-Strategien benötigt werden. Die entwickelten Mapping-Strategien sowie die darauf basierenden Methoden zur Modelltransformation werden in den folgenden Unterkapiteln hergeleitet.

7.1 Mapping der Modellelemente

Um eine Transformation von Simulationsmodellen durchführen zu können, müssen zunächst die extrahierten Modellelemente eines Simulationsmodells des gewählten Ausgangswerkzeugs auf die entsprechenden Modellelemente eines Simulationsmodells des gewünschten Zielwerkzeugs abgebildet werden. Dieses Vorgehen wird **Mapping** genannt. Dabei wird zwischen 1:1-Mapping und komplexen Mapping unterschieden (Bild 7-2).

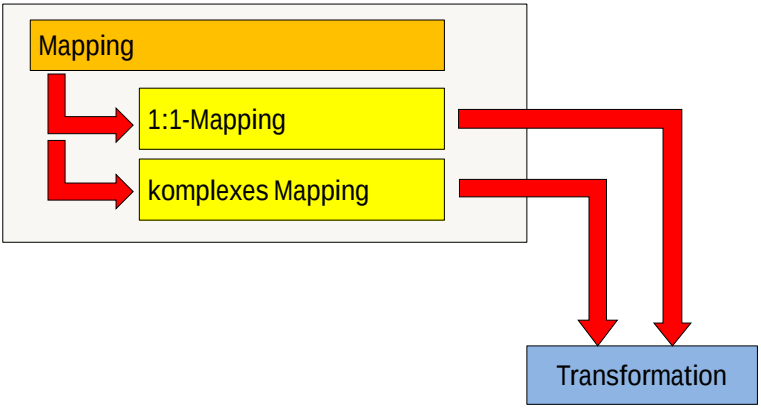


Bild 7-2: Mapping als Voraussetzung für die Transformation

Ergebnis eines Mappings ist also die Kenntnis darüber, welchem Modellelement des Zielwerkzeugs ein Modellelement des Ausgangswerkzeugs entspricht (Bild 7-3). Bei blockschaltbildbasierten Simulationswerkzeugen (wie z.B. Matlab/Simulink und Virtuos) entspricht ein Block einem Modellelement. Die konkrete Transformation kann auf Basis der Mapping-Strategien durchgeführt werden.

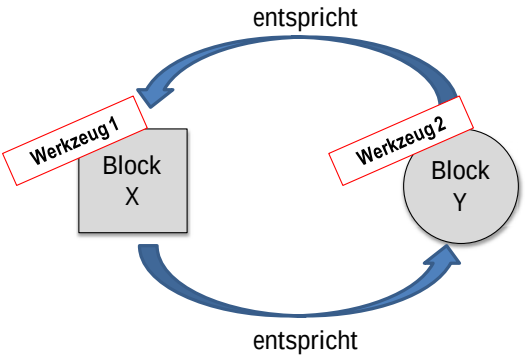


Bild 7-3: Ergebnis eines Mappings

7.1.1 Definitionen

1:1-Mapping

Beim 1:1-Mapping handelt es sich um eine Methode zum **direkten** Vergleich und zur Herstellung der Beziehung zwischen **gleichartigen** Modellelementen aus verschiedenen Simulationswerkzeugen. Die Eingänge, Ausgänge und Parameter stimmen bei Ausgangs- und Zielwerkzeug exakt überein, so dass sie direkt aufeinander abgebildet werden können.

Als Beispiel für ein 1:1-Mapping soll ein einfacher Additionsblock dienen. Dieser Block funktioniert unabhängig vom verwendeten Simulationswerkzeug immer nach dem gleichen Prinzip: Die zwei Eingänge werden addiert, am Ausgang wird die Summe der beiden Eingänge gebildet (Bild 7-4).

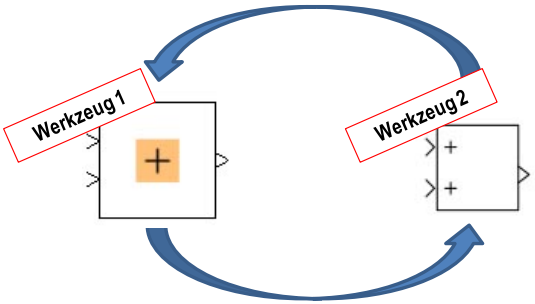


Bild 7-4: 1:1-Mapping

Komplexes Mapping

Ein komplexes Mapping ist eine Methode zum Vergleich und Herstellung der Beziehung zwischen **abweichenden** und **nicht vorhandenen** Modellelementen aus verschiedenen Simulationswerkzeugen. In diesem Fall weichen die Eingänge, Ausgänge und Parameter voneinander ab oder sind in einem Simulationswerkzeug nicht vorhanden. Das komplexe Mapping muss für alle Spezialelemente angewendet werden.

Sowohl beim 1:1-Mapping als auch beim komplexen Mapping wird zwischen äquivalenten und ähnlichen Modellelementen unterschieden. Diese beiden Begriffe werden im Folgenden definiert.

Äquivalentes Modellelement

Ein Modellelement E2 sei zu einem Modellelement E1 äquivalent, wenn sowohl die Anzahl als auch die Art der Ein- und Ausgänge sowie die Anzahl und Art der Parameter identisch sind und sich das Verhalten von E1 und E2 durch den Zusammenhang $f_2(\underline{x},\underline{p}_2) = f_1(\underline{x},\underline{p}_1)$ beschreiben lässt. Das logische Verhalten eines Modellelements E1 sei durch die Funktion $f_1(\underline{x},\underline{p}_1)$, das logische Verhalten eines Modellelements E2 sei durch die Funktion $f_2(\underline{x},\underline{p}_2)$ definiert. Das Verhalten eines Modellelements entspricht der mathematischen Abbildung von dessen Eingängen $(\underline{x},\underline{p})$ auf dessen Ausgänge $f(\underline{x},\underline{p})$, also dessen mathematischer Übertragungsfunktion. Mit $\underline{p}$ seien die Parameter eines Modellelements beschrieben.

Ähnliches Modellelement

Ein Modellelement E2 sei zu einem Modellelement E1 ähnlich, wenn zum einen die Anzahl und Art der Ein- und Ausgänge identisch sind und sich zum anderen das Verhalten von E1 und E2 durch den Zusammenhang $f_2(\underline{x},\underline{p}_2) = k_1 \cdot f_1(\underline{x}+\underline{x}_{0,}\ ,\underline{p}_1) + k_2$ abbilden lässt. Analog ist ein Modellelement E3 zu einem Modellelement E2 ähnlich, wenn zum einen die Anzahl und Art der Ein- und Ausgänge identisch sind und sich zum anderen das Verhalten von E2 und E3 durch den Zusammenhang $f_3(\underline{x},\underline{p}_3) = k_1 \cdot f_2(\underline{x}+\underline{x}_{0,}\ ,\underline{p}_2) + k_2$ abbilden lässt. Dabei seien k_1 und k_2 Konstanten. Handelt es sich bei den ähnlichen Modellelementen um Standardelemente, so seien sie als **ähnliche Standardelemente** definiert.

In Kapitel 7.1.3 wird die Vorgehensweise beim Mapping im ersten Schritt vom Ausgangswerkzeug zum Meta-Metamodell zunächst in einem graphischen Vorgehensmodell dargestellt und dann im Detail erläutert. Ebenso wird das Vorgehen beim Mapping im zweiten Schritt vom Meta-Metamodell zum Zielwerkzeug in Kapitel 7.1.4 zuerst graphisch zusammengefasst und anschließend detailliert. Zum besseren Verständnis folgen drei Beispiele.

7.1.2 Allgemeine Methode zur Durchführung eines Mappings

E1 sei ein Modellelement im Ausgangswerkzeug. E2 sei ein zu E1 äquivalentes Modellelement im Meta-Metamodell. E3 sei ein zu E1 und E2 äquivalentes Modellelement im Zielwerkzeug. Da bei der Konzeption der Wiederverwendung die Vorgehensweise der indirekten Modelltransformation über ein Metamodell gewählt wurde, muss das Mapping in zwei Schritten erfolgen. Bevor ein Mapping eines Modellelements E1 im Ausgangswerkzeug auf ein äquivalentes Modellelement E3 im Zielwerkzeug erfolgen kann, muss zunächst ein Mapping von E1 auf ein äquivalentes Modellelement im Meta-Metamodell E2 durchgeführt werden. Erst im zweiten Schritt kann dann ein Mapping von E2 auf E3 erfolgen. In Bild 7-5 ist diese allgemeine Methode zur Durchführung eines Mappings in zwei Schritten schematisch dargestellt.

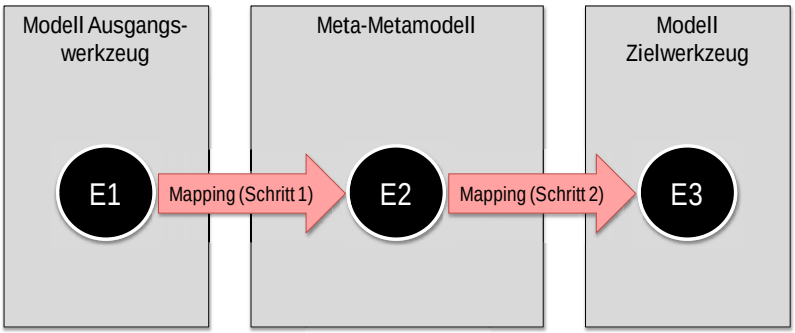


Bild 7-5: Allgemeine Methode zur Durchführung eines Mappings

Die konkrete Methode zur Durchführung der Mappings in Schritt 1 und Schritt 2 wird in den folgenden Unterkapiteln definiert. In Kapitel 7.1.3 wird die Methode mit der konkreten Vorgehensweise für den ersten Schritt - das Mapping von Modellelementen im Ausgangswerkzeug auf äquivalente Modellelemente im Meta-Metamodell - definiert und in Bild 7-6 graphisch dargestellt.

7.1.3 Schritt 1: Mapping vom Ausgangswerkzeug zum Meta-Metamodell

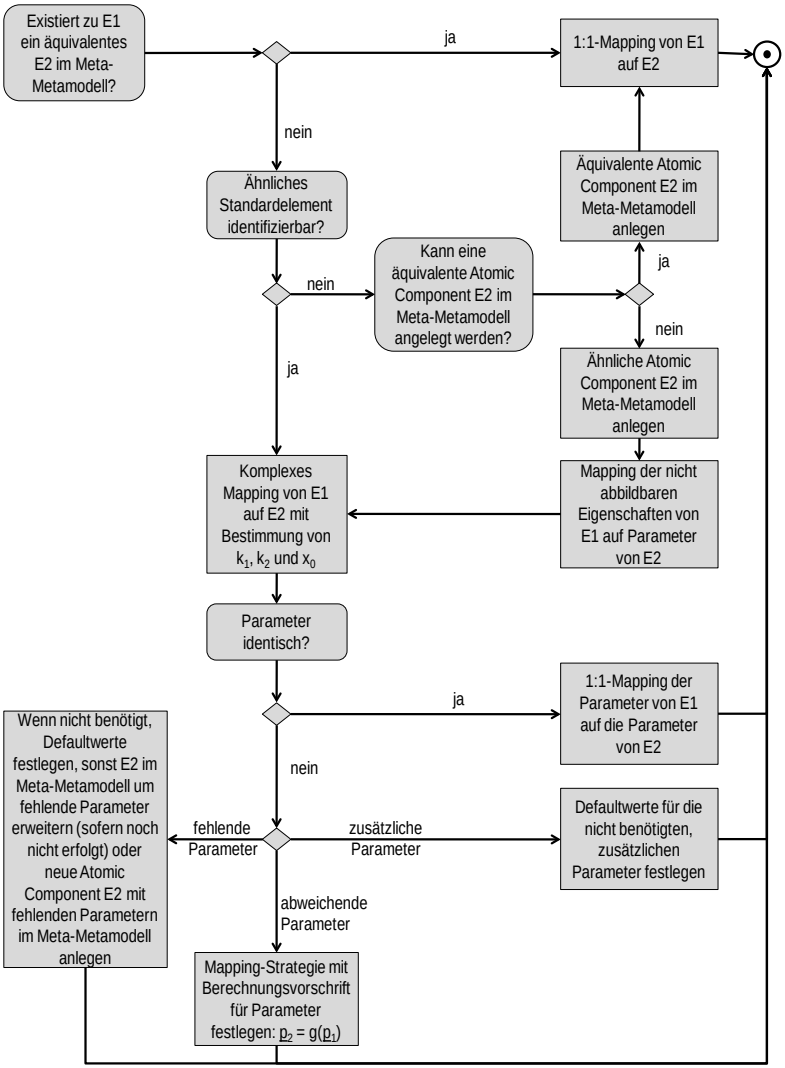


Bild 7-6: Mapping-Strategien für das Mapping von Modellelementen im Ausgangswerkzeug auf Modellelemente im Meta-Metamodell

Soll ein Mapping eines Modellelements E1 eines Ausgangswerkzeugs auf ein Modellelement E2 des Meta-Metamodells erfolgen, wird zunächst geprüft, ob zu E1 bereits ein äquivalentes Modellelement E2 im Meta-Metamodell existiert. Ist dies der Fall, kann direkt ein 1:1-Mapping von E1 auf E2 durchgeführt werden. Dies ist der einfachste Fall eines Mappings. Ist kein zu E1 äquivalentes Modellelement E2 im Meta-Metamodell vorhanden, wird geprüft, ob im Meta-Metamodell ein zu E1 ähnliches Standardelement existiert. Ist dies nicht der Fall, wird geprüft, ob im Meta-Metamodell ein zu E1 äquivalentes Modellelement E2 als neue *Atomic Component* angelegt werden kann. Besteht diese Möglichkeit, wird ein zu E1 äquivalentes Modellelement E2 als neue *Atomic Component* im Meta-Metamodell angelegt. Danach kann wiederum ein 1:1-Mapping von E1 auf E2 erfolgen. Ist es nicht möglich, ein äquivalentes Modellelement E2 als *Atomic Component* im Meta-Metamodell anzulegen, wird ein ähnliches Modellelement E2 als *Atomic Component* im Meta-Metamodell definiert. Für die nicht abbildbaren Eigenschaften von E1 muss dann ein komplexes Mapping dieser Eigenschaften auf zu definierende Parameter von E2 erfolgen.

Ist ein zu E1 ähnliches Standardelement E2 im Meta-Metamodell bereits vorhanden, so kann ein komplexes Mapping von E1 auf E2 erfolgen. Bei einem ähnlichen Standardelement handelt es sich um ein Standardelement, das die gleiche Anzahl von Ein- und Ausgängen wie E1 besitzt und dessen Verhalten mit dem Verhalten von E1 in den Zusammenhang

$$f_2(\underline{x},\underline{p}_2) = k_1 \cdot f_1(\underline{x}+\underline{x}_0, ,\underline{p}_1)+ k_2$$

gebracht werden kann. Zur Durchführung des komplexen Mappings müssen die Konstanten k_1 und k_2 sowie $\underline{x}_0$ mathematisch bestimmt werden.

Schließlich ist noch zu überprüfen, ob die Parameter der Modellelemente E1 und E2 identisch sind. Ist dies der Fall, müssen keine weiteren Mapping-Strategien definiert werden, das komplexe Mapping ist abgeschlossen. Bei einer Abweichung der Parameter, können drei Fälle auftreten. Die jeweilige Methode des Mappings wird für die drei Fälle beschrieben und anhand von Beispielen demonstriert.

1. Abweichende Parameter

Der erste Fall setzt voraus, dass die Parameter prinzipiell übereinstimmen und nur geringfügig voneinander abweichen, indem zum Beispiel eine andere Einheit verwendet wird. In diesem Fall muss das Modellelement E2 im Meta-Metamodell nicht modifiziert werden, sondern es muss lediglich eine neue Mapping-Strategie für diesen Parameter festgelegt werden. In dieser wird die Berechnungsvorschrift $p_2 = g(p_1)$ zur Umrechnung des entsprechenden Parameters umgesetzt, die schließlich ein komplexes Mapping von E1 auf E2 ermöglicht.

Die methodische Vorgehensweise bei einem komplexen Mapping bei einem oder mehreren abweichenden Parametern soll am Beispiel eines Sinus-Modellelements in zwei verschiedenen Simulationswerkzeugen verdeutlicht werden. In dem betrachteten Fall ist das Sinus-Modellelement sowohl in Ausgangs- als auch in Zielwerkzeug als Block vorhanden. Ein 1:1-Mapping ist jedoch nicht möglich, da einzelne Parameter voneinander abweichen. Der Sinus-Block ist in beiden im Beispiel betrachteten Simulationswerkzeugen eine Quelle, die eine Sinusschwingung in ein Simulationsmodell einbringt. Dieser Block hat in seiner Funktion als Quelle keinen Eingang sondern nur einen Ausgang, an dem die im Block parametrierte Sinusschwingung ausgegeben wird. Darüber hinaus liegen in beiden Simulationswerkzeugen spezielle Eigenschaften vor, die sich voneinander unterscheiden. In dem betrachteten Beispiel weicht unter anderem der Parameter, der die Periode der Sinusschwingung angibt, im Ausgangswerkzeug von dem im Zielwerkzeug ab. Im Ausgangswerkzeug wird der Parameter „Frequency“ angegeben, während das Zielwerkzeug den Parameter „Period“ verwendet, wie in Bild 7-7 zu sehen ist.

Ausgangswerkzeug				Meta-Metamodell			Zielwerkzeug			
Sin (E1)				SineWave (E2)			Sine (E3)			
	Inputs	Outputs	Parameter	Inputs	Outputs	Parameter		Inputs	Outputs	Parameter
	-/-	1	Amplitude	-/-	out	Amplitude		-/-	out	Magnitude
			Frequency			Frequency				
										Period
			Phase			Phase				XDisplacement
			Bias			Bias				YDisplacement
			Sine Type							
			Time							

f = 1/T

Bild 7-7: Beispiel komplexes Mapping (abweichende Parameter)

Beim ersten Schritt des Mappings vom Ausgangswerkzeug zum Meta-Metamodell wird zunächst geprüft, ob ein äquivalentes Modellelement im Meta-Metamodell vorhanden ist. Ein Sinus-Element ist ein Standardelement, das im WieMod Meta-Metamodell vorhanden ist und dort 0 Eingänge, 1 Ausgang und die Parameter „Amplitude“, „Frequency“, „Phase“ und „Bias“ besitzt. Im Ausgangswerkzeug hat das Sinus-Modellelement den Namen „Sin“, das entsprechende Modellelement im Meta-Metamodel heißt „SineWave“. Die Ein- und Ausgänge des Modellelements E1 stimmen mit denen des Modellelements im Meta-Metamodell E2 überein. Die Anzahl und Art der Parameter weicht jedoch voneinander ab. Aus diesem Grund ist das Sinus-Modellelement im Meta-Metamodell E2 lediglich als zu E1 ähnliches Standardelement zu identifizieren. Somit muss keine neue *Atomic Component* im Meta-Metamodell angelegt werden, sondern eine Mapping-Strategie von E1 auf E2 muss festgelegt werden. Das Verhalten der Modellelemente kann mit $f_2(\underline{x},\underline{p}_2) = \sin x = f_1(\underline{x},\underline{p}_1)$, mit $k_1=1$, $k_2=0$ und $\underline{x}_0=0$ beschrieben werden. E1 hat im Vergleich zu E2 zwei zusätzliche Parameter „SineType“ und „Time“, die anderen Parameter sind identisch. Für die zusätzlichen Parameter werden Defaultwerte definiert, für die vier übereinstimmenden Parameter erfolgt ein 1:1-Mapping. Somit kann das Mapping vom Ausgangswerkzeug zum Meta-Metamodell durchgeführt werden.

Das methodische Vorgehen beim Mapping vom Meta-Metamodell zum Zielwerkzeug erfolgt auf die gleiche Weise. Zuerst wird geprüft, ob ein zu E2 äquivalentes Modellelement E3 im Zielwerkzeug existiert. Das Sinus-Modellelement im Zielwerkzeug hat den Namen „Sine“. Dessen Ein- und Ausgänge stimmen mit denen von E2 überein, die Parameter weichen jedoch voneinander ab. Somit ist E3 zu E2 kein äquivalentes, sondern ein ähnliches Modellelement. Das Verhalten der Modellelemente kann mit $f_2(\underline{x},\underline{p}_2) = \sin x = f_3(\underline{x},\underline{p}_3)$, mit $k_1=1$, $k_2=0$ und $\underline{x}_0=0$, beschrieben werden. Das Modellelement E3 hat die Parameter „Magnitude“, „Period“, „XDisplacement“ und „YDisplacement“. Der Parameter „Magnitude“ von E3 entspricht genau dem Parameter „Amplitude“ von E2. Die Namen sind zwar unterschiedlich, die Parameter sind jedoch gleich. Ebenso entspricht der Parameter „XDisplacement“ von E3 genau dem Parameter „Phase“ von E2, der Parameter „YDisplacement“ von E3 entspricht genau dem Parameter „Bias“ von E2. Für diese Parameter kann somit ein 1:1-Mapping erfolgen. Dies trifft für die Parameter „Period“ in E3 und „Frequency“ in E2 jedoch nicht zu. Periode und Frequenz bilden den gegenseitigen Kehrwert und hängen über die einfache Rechenvorschrift $f=1/T$ voneinander ab. In diesem Fall muss eine neue Mapping-Strategie definiert werden, in der diese Rechenvorschrift als Zusatzinformation hinterlegt ist. Für die gege-

benen Parameter $p_2=f$, der die Frequenz der Sinusschwingung beschreibt und $p_3=T$, der die Schwingungsdauer beschreibt, gilt der Zusammenhang $p_3=1/p_2$. Erst nach Definition dieser Mapping-Strategie kann das Mapping vom Meta-Metamodell zum Zielwerkzeug durchgeführt werden.

2. Fehlende Parameter

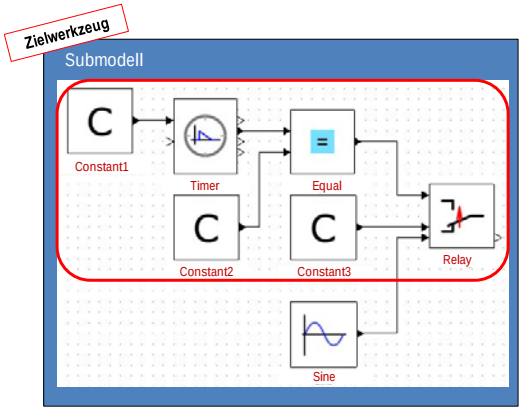
Fehlen dem Modellelement E2 im Vergleich zum Modellelement E1 ein oder mehrere Parameter, können verschiedene Strategien angewendet werden. Werden die Parameter in E2 im Hinblick auf das entsprechende Element im Zielwerkzeug E3 nicht benötigt, werden für diese Defaultwerte so definiert, dass diese beim Mapping nicht berücksichtigt werden. Ansonsten können abhängig von der Anzahl und Art der fehlenden Parameter diese einerseits zu dem bestehenden Modellelement E2 im Meta-Metamodell hinzugefügt werden. Für den Fall, dass bestehende Mappings bereits darauf zugreifen und die neu eingeführten Parameter nicht benötigen, müssen diese auch mit Defaultwerten versehen werden, so dass bestehende Mappings von den neuen Parametern nicht beeinflusst werden. Die andere Möglichkeit besteht darin, eine neue *Atomic Component* im Meta-Metamodell zu definieren, die die bislang fehlenden Parameter direkt beinhaltet. Beide alternativen Möglichkeiten sind im Einzelfall hinsichtlich des zur Umsetzung erforderlichen Aufwands gegeneinander abzuwägen. Beide ermöglichen gleichermaßen die Durchführung eines komplexen Mappings von E1 auf E2.

Für den Fall, dass im Meta-Metamodell im Vergleich zum Ausgangswerkzeug Parameter fehlen, diese jedoch benötigt werden, ist es nicht ausreichend, diese lediglich mit Defaultwerten zu belegen. In dem Beispiel des Sinus-Blocks besteht in einem Simulationswerkzeug die Möglichkeit, über den Parameter „Time“ die Schwingungsdauer der Sinusschwingung einzustellen. Dieser Parameter ist jedoch nicht in allen Simulationswerkzeugen vorgesehen (Bild 7-8). Soll dieser jedoch verwendet werden, kann er entweder zu dem bestehenden Modellelement E2 im Meta-Metamodell hinzugefügt werden oder eine neue *Atomic Component* muss im Meta-Metamodell definiert werden, die den fehlenden Parameter direkt beinhaltet. Im betrachteten Beispiel ist der einfachere Weg, den Parameter zu dem bestehenden Modellelement „SineWave“ hinzuzufügen. Im ersten Schritt des Mappings kann dann ein 1:1-Mapping des bestehenden Parameters „Time“ von E1 direkt auf den neu angelegten Parameter „Time“ von E2 erfolgen.

Ausgangswerkzeug				Meta-Metamodell			Zielwerkzeug			
Sin (E1)				SineWave (E2)			Sine (E3)			
	Inputs	Outputs	Parameter	Inputs	Outputs	Parameter		Inputs	Outputs	Parameter
	-/-	1	Amplitude	-/-	out	Amplitude		-/-	out	Magnitude
			Frequency			Frequency				
										Period
			Phase			Phase				XDisplacement
			Bias			Bias				YDisplacement
			Sine Type							
			Time			Time				

Bild 7-8: Beispiel komplexes Mapping (fehlender Parameter)

Im zweiten Schritt des Mappings kann der im Zielwerkzeug im Vergleich zum Meta-Metamodell fehlende Parameter als Submodell generiert werden. Dazu muss das benötigte Submodell, mit dem der fehlende Parameter nachgebildet werden kann, zunächst einmalig im Zielwerkzeug manuell erstellt werden. In Bild 7-9 ist zu sehen, wie dies mit Hilfe von im Zielwerkzeug vorhandenen Modellelementen beispielhaft der fehlenden Parameter „Time“ erfolgen kann. Diese Kombination von Modellelementen kann als Submodell im Zielwerkzeug hinterlegt werden. Bei einem komplexen Mapping von E2 auf E3 wird dieses Submodell aufgerufen und mit dem jeweiligen Wert des Parameters „Time“ ergänzt und auf diese Weise automatisch generiert.



(Beispiel: Parameter „Time“ bei Sinus-Block)

Bild 7-9: Fehlender Parameter als Submodell

3. Zusätzliche Parameter

Enthält das Modellelement E2 im Vergleich zum Modellelement E1 einen oder mehrere zusätzliche Parameter, die jedoch von E1 nicht benötigt werden, müssen diese mit Defaultwerten belegt werden, so dass diese ohne Einfluss bleiben. Auf diese Weise kann ein komplexes Mapping von E1 auf E2 trotz überzähliger Parameter erfolgen.

Der erste Schritt der Durchführung eines Mappings vom Ausgangswerkzeug zum Meta-Metamodell ist somit abgeschlossen. Im folgenden zweiten Schritt wird schließlich ein Mapping vom Meta-Metamodell zum Zielwerkzeug durchgeführt. Die Methode mit der konkreten Vorgehensweise wird in Kapitel 7.1.4 hergeleitet und in Bild 7-10 graphisch dargestellt.

7.1.4 Schritt 2: Mapping vom Meta-Metamodell zum Zielwerkzeug

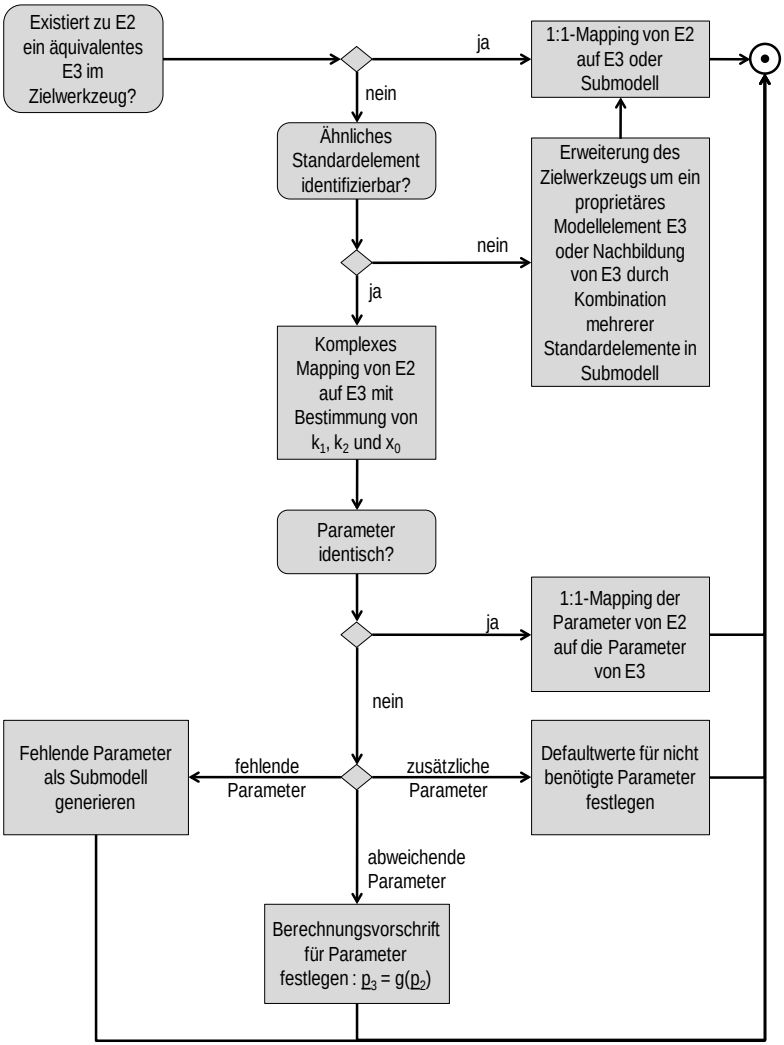


Bild 7-10: Mapping-Strategien für das Mapping von Modellelementen im Meta-Metamodell auf Modellelemente im Zielwerkzeug

Die Methode beim zweiten Schritt der Durchführung eines Mappings vom Meta-Metamodell zum Zielwerkzeug ist im Wesentlichen gleich der Methode beim ersten Schritt. Lediglich zwei Fälle weichen voneinander ab. Ist im Zielwerkzeug kein zum Modellelement E2 im Meta-Metamodell ähnliches Standardelement E3 vorhanden, existieren zwei Möglichkeiten. Zum einen ist es bei einigen Simulationswerkzeugen möglich, proprietäre, benutzerdefinierte Modellelemente zu definieren. In diesem Fall kann das fehlende Modellelement E3 im Zielwerkzeug angelegt werden, so dass ein 1:1-Mapping von E2 auf E3 erfolgen kann. Ist das Hinzufügen benutzerdefinierter Modellelemente im Zielwerkzeug nicht möglich, besteht die Möglichkeit, das fehlende Modellelement durch eine Kombination bestehender Standardelemente im Zielwerkzeug nachzubilden und diese als Submodell zu generieren. In diesem Fall ist ein komplexes Mapping von E2 auf das generierte Submodell notwendig.

Die Methode zur Vorgehensweise bei abweichenden und zusätzlichen Parametern ist die gleiche wie bei Schritt 1. Fehlen dem Modellelement E3 im Vergleich zum Modellelement E2 ein oder mehrere Parameter, müssen diese analog zu der Vorgehensweise bei einem fehlenden Modellelement in einem Submodell generiert werden. So kann wiederum ein komplexes Mapping der Parameter des Modellelements E2 auf die des Modellelements E3 durchgeführt werden.

4. Fehlendes Modellelement

Die nächste Ausbaustufe besteht darin, ein komplexes Mapping nicht nur für abweichende oder fehlende Parameter vorzunehmen, sondern auch dann ein Mapping zu ermöglichen, wenn ganze Blöcke in einem der verwendeten Simulationswerkzeuge fehlen. Diese Problematik soll am Beispielmodell (Bild 7-11) mit drei Konstanten, die mit einem gemischten Additions-/Subtraktions-Block verbunden sind, erläutert werden. Die Vorzeichenreihenfolge des gemischten Additions-/Subtraktions-Blocks lautet „+ - +“. An dem Ausgang des Blocks werden die Werte der Konstanten addiert bzw. subtrahiert.

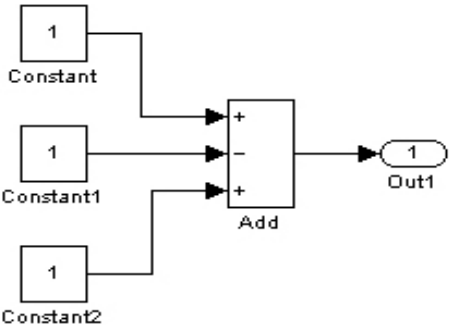


Bild 7-11: Modell mit gemischtem Additions-/Subtraktions-Block

Viele Simulationswerkzeuge verwenden keine Blöcke mit mehr als zwei Eingängen. Ebenso werden oftmals keine Blöcke mit gemischten Vorzeichen unterstützt. In einigen Simulationswerkzeugen stellt es eine zusätzliche Problematik dar, wenn der erste Eingang subtrahiert werden soll.

Für das oben vorgestellte Beispielmodell kann ein 1:1-Mapping nicht durchgeführt werden, da kein zu dem Modellelement E1, in diesem Beispiel der gemischte Addition-/Subtraktionblock, äquivalentes Modellelement im Meta-Metamodell E2 existiert. Ein ähnliches Standardelement ist im Meta-Metamodell ebenfalls nicht vorhanden, so dass überprüft werden muss, ob das Anlegen eines äquivalenten Modellelements E2 im Meta-Metamodell möglich ist. Da das Anlegen vorzeichenbehafteter Eingänge für ein Modellelement im Meta-Metamodell nicht möglich ist, muss ein ähnliches Modellelement E2 im Meta-Metamodell definiert werden. Für die nicht abbildbaren Eigenschaften von E1 wird ein komplexes Mapping dieser Eigenschaften auf einen zu definierenden Parameter von E2 durchgeführt. Im Beispiel wird eine neue *Atomic Component* „AddSubMix“ im Meta-Metamodell angelegt. Zusätzlich wird der Parameter „InputSignums“ definiert. In einem komplexen Mapping von E1 auf E2 („AddSubMix“) wird die Folge der Vorzeichen von E1 auf den Parameter „InputSignums“ gemappt. Das Verhalten von E1 und E2 kann durch $f_1(\underline{x}) = x_1-x_2+x_3 = f_2(\underline{x})$, mit $k_1=1$, $k_2=0$ und $\underline{x}_0=0$ beschrieben werden. Abschließend wird geprüft, ob die Parameter von E1 und E2 insgesamt identisch sind. Da E1 gegenüber E2 der Parameter „InputSignums“ fehlt, muss E2 im Meta-Metamodell um den fehlenden Parameter erweitert werden. Dies ist bei der Definition der *Atomic Component* „AddSubMix“ bereits erfolgt. Somit ist Schritt 1 des Mappings,

nämlich von dem Modellelement E1 des Ausgangswerkzeugs auf ein Modellelement E2 im Meta-Metamodell, abgeschlossen.

Im zweiten Schritt des Mappings, nämlich von dem Modellelement E2 im Meta-Metamodell (in diesem Fall „AddSubMix“), auf ein Modellelement E3 im Zielwerkzeug, muss zunächst wieder überprüft werden, ob ein zu E2 äquivalentes E3 bereits vorhanden ist. Da dies nicht der Fall ist, wird geprüft, ob ein zu E2 ähnliches E3 existiert. Dies ist ebenfalls nicht zutreffend. Damit das Modellelement E2 universell für mehrere unterschiedliche Zielwerkzeuge nutzbar ist, wird kein proprietäres Modellelement im Zielwerkzeug definiert, sondern die Variante der Nachbildung von E3 durch die Kombination mehrerer Standardelemente in einem Submodell gewählt. So kann schließlich ein komplexes Mapping von „AddSubMix“ auf ein automatisch generiertes Submodell im Zielwerkzeug vorgenommen werden. Die gesamte Methode zur Vorgehensweise bei fehlenden Modellelementen ist in Bild 7-12 zusammenfassend graphisch dargestellt.

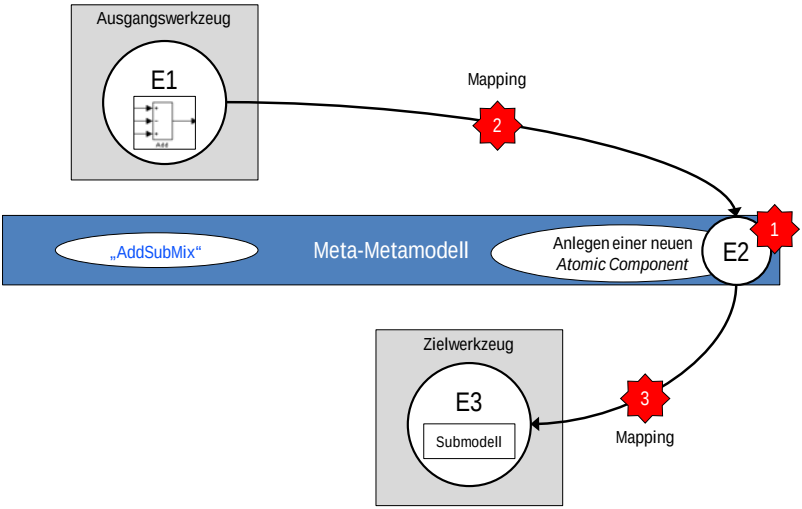


Bild 7-12: Mapping-Strategie bei nicht vorhandenen Modellelementen

Das für das Zielwerkzeug automatisch generierte Submodell besteht aus einem Additions-Block, der mit einem Subtraktions-Block verbunden ist (Bild 7-13). Um die Vorzeichen des gemischten Additions-/Subtraktions-Blocks mappen zu können, wurde eine

Methode entwickelt, mit der ein generisches Mapping für diese Modellelemente erfolgen kann. Dabei werden diese in Arrays am Additions- bzw. Subtraktions-Block des Submodells gespeichert. Da der erste und dritte Eingang des gemischten Blocks positiv sind, wird am Additions-Block das Array [1,3] gespeichert. Der zweite Eingang ist negativ, so dass am Subtraktions-Block das Array [2] gespeichert wird. Die Ausgänge der Konstanten werden anhand dieser Arrays automatisch auf den richtigen Eingang des Additions- bzw. Subtraktions-Block im Submodell gegeben. Mit dieser Methode kann generisch ein Mapping für diesen Typ nicht vorhandener Blöcke mit beliebiger Vorzeichenzahl und -reihenfolge erfolgen.

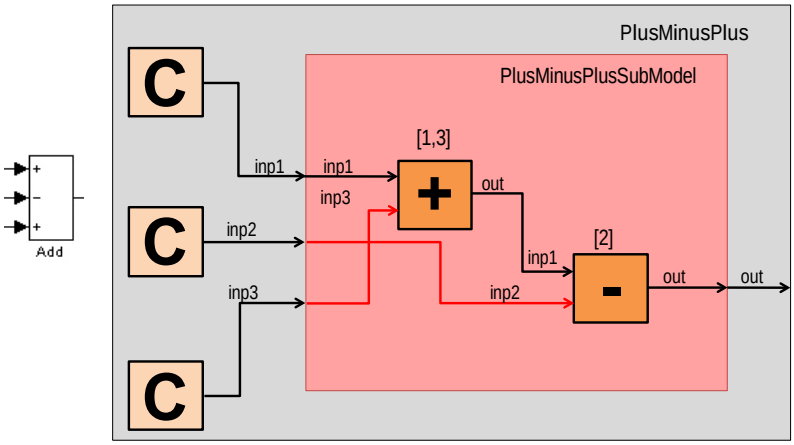


Bild 7-13: Generiertes Submodell für den gemischten Additions-/Subtraktions-Block

Analog erfolgt das komplexe Mapping für gemischte Multiplikations-/Divisions-Blöcke. Die neu angelegte *Atomic Component* im Metamodell hat analog den Namen „MultDivMix“. Die konkrete Umsetzung eines komplexen Mappings wird in Kapitel 8.1.6 am Beispiel des gemischten Additions-/Subtraktions-Blocks beschrieben.

7.2 Transformation der Modelle

Die Transformation ganzer Modelle erfolgt analog zur Durchführung von Mappings in zwei Schritten: Im ersten Schritt wird das Modell eines Ausgangswerkzeugs in das Meta-Metamodell transformiert. Im zweiten Schritt wird das Modell aus dem Meta-Metamodell in ein Modell für das gewünschte Zielwerkzeug transformiert. (Bild 7-14)

Aufgrund der Tatsache, dass beim Hinzufügen eines neuen Werkzeugs immer zwei Schnittstellen, nämlich eine vom neuen Werkzeug zum Metamodell und eine vom Metamodell zum neuen Werkzeug, geschaffen werden, ist immer eine bidirektionale Transformation möglich. Sobald ein Simulationsmodell eines beliebigen Ausgangswerkzeugs eingelesen wird, kann somit immer eine Transformation in jedes andere integrierte Zielwerkzeug erfolgen. Prinzipiell wird bei der Modelltransformation ein Modell des Ausgangswerkzeugs eingelesen und in das Meta-Metamodell überführt. Auf Basis der Mapping-Strategien werden die Modellelement des Ausgangswerkzeugs durch die entsprechenden Modellelemente oder Submodelle, die eine Kombination mehrerer Modellelemente des Zielwerkzeugs enthalten, ersetzt. Dieser Vorgang des Austauschens der Modellelemente entsprechend der zuvor definierten Mapping-Strategien wird als Transformation bezeichnet.

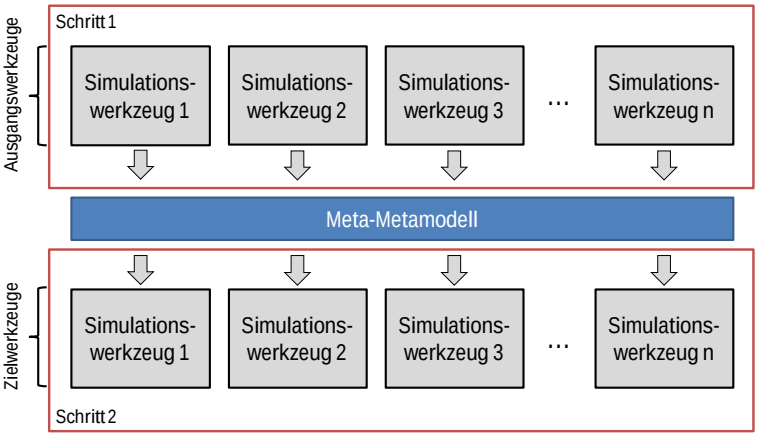


Bild 7-14: Schritte der Modelltransformation

7.2.1 Schritt 1: Modelltransformation vom Ausgangswerkzeug zum Meta-Metamodell

Bild 7-15 veranschaulicht den ersten Schritt der Transformation ganzer Modelle eines Ausgangswerkzeugs in das Meta-Metamodell auf Basis der zuvor definierten Mapping-Strategien. Zur Vereinfachung und anschaulicheren Darstellung werden beispielhaft nur 1:1-Mappings verwendet. Dargestellt ist, wie die eigentliche Transformation von Modellen als Anwendung der Mapping-Strategien erfolgt und die Modellelemente des Ausgangswerkzeugs $E1_x$ dabei durch die entsprechenden Modellelemente des Meta-Metamodells $E2_x$ ersetzt werden.

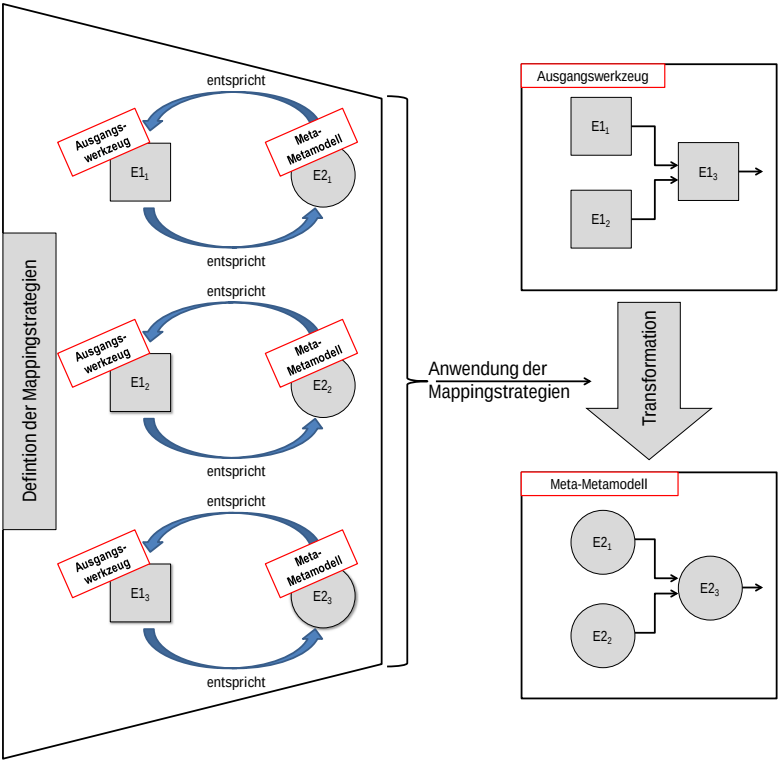


Bild 7-15: Schritt 1 der Transformation eines Modells eines Ausgangswerkzeugs zum Meta-Metamodell als Anwendung der Mapping-Strategien

7.2.2 Schritt 2: Modelltransformation vom Meta-Metamodell zum Zielwerkzeug

Der zweite Schritt der Modelltransformation vom Meta-Metamodell zum Zielwerkzeug erfolgt analog zum ersten Schritt, der in Kapitel 7.2.1 entwickelt wurde. Hierbei werden wieder die zuvor definierten Mapping-Strategien der Modellelemente des Meta-Metamodells auf die entsprechenden Modellelemente des Zielwerkzeugs angewandt. Dabei werden Modellelemente des Meta-Metamodells $E2_x$ durch die entsprechenden Modellelemente des Zielwerkzeugs $E3_x$ oder eine Kombination mehrerer Modellelemente des Zielwerkzeugs, die in einem Submodell zusammengefasst sind, ersetzt. Dies ist in Bild 7-16, wieder in vereinfachter Form, dargestellt.

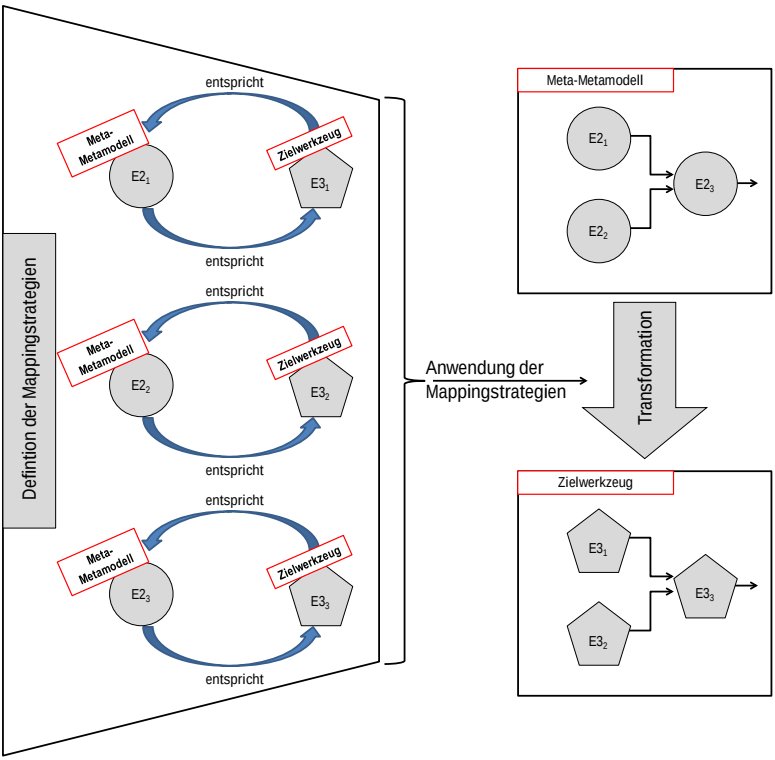


Bild 7-16: Schritt 2 der Transformation eines Modells eines Ausgangswerkzeugs zum Meta-Metamodell als Anwendung der Mapping-Strategien

8 Realisierung

In diesem Kapitel werden die prototypische Umsetzung des in dieser Arbeit und im Projekt „WieMod“ entwickelten Konzepts zur Wiederverwendung von Simulationsmodellen sowie die zu dessen Anwendung benötigten Methoden beschrieben. Die Implementierung basiert auf der Entwicklungsumgebung Eclipse und wird im weiteren Verlauf dieser Arbeit als WieMod-Workbench bezeichnet. Abschließend werden Konzept und Methoden anhand eines ausgewählten Simulationsmodells validiert.

8.1 Implementierung

8.1.1 Entwicklungsumgebung Eclipse

Eclipse wurde 2001 von IBM, Object Technology International (OTI) und acht weiteren Firmen ins Leben gerufen. Das Eclipse-Projekt besteht aus drei Unterprojekten: Die Eclipse-Plattform selbst bildet das Rückgrat der ganzen Anwendung. Das Java Development Toolkit (JDT) umfasst die Java-Entwicklungs-Tools. Im Plug-In Development Environment (PDE) können eigene Plug-Ins für Eclipse entwickelt werden. /60/

Eclipse ist eine quelloffene integrierte Entwicklungsumgebung (Integrated Development Environment, IDE), die durch Plug-Ins beliebig erweitert und somit individuell für unterschiedlichste Entwicklungsaufgaben angepasst werden kann. /61/ Die Eclipse-Plattform ist in Java implementiert, Plug-Ins für Eclipse werden ebenfalls in Java entwickelt. Auch das PDE selber ist ein Eclipse Plug-In. Jedes Unterprojekt besteht wieder aus verschiedenen Unterprojekten. So enthält ein JDT-Unterprojekt beispielsweise die Unterprojekte User Interface (UI, Benutzerschnittstellen), Core (Kern) und Debug. /60/

In seiner Funktion als Java-Entwicklungsumgebung unterstützt Eclipse Programmierer bei der effizienten Entwicklung in Java, einer objektorientierten Programmiersprache. Dabei unterstützen die Java-Entwicklungswerkzeuge in Eclipse den Entwicklungsprozess durch die Automatisierung trivialer und zeitraubender Operationen. Weitere Tools in Eclipse ermöglichen die Generierung und Bearbeitung von Java-Code sowie die Navigation darin. So bietet der Java-Editor Syntaxhervorhebung, Codevervollständigung, Codeassistenz sowie mehrere sich ergänzende Ansichten, die einen Überblick über die Codestruktur und noch offene Aufgaben bieten. Zusätzlich beinhaltet Eclipse einen voll

ausgestatteten Debugger, mit dem Haltepunkte gesetzt, Variablenwerte angezeigt und modifiziert werden können. /62/

Eclipse ist jedoch nicht nur eine integrierte Entwicklungsumgebung (IDE), sondern bietet auch eine Plattform für die Tool-Integration und bringt für eine effiziente Arbeitsweise Tools für den Entwurf, die Programmierung und das Testen zusammen. So können alle Mitglieder eines Entwicklungsteams auf einheitliche Art und Weise Ressourcen erstellen, verändern und verwalten. Mit Hilfe seiner Architektur (Bild 8-1) und einer Reihe von Java-Frameworks erleichtert Eclipse das Entwickeln integrierter Tools. Durch die Integration vorhandener sowie eigener Plug-Ins kann die Plattform beliebig erweitert werden. /62/

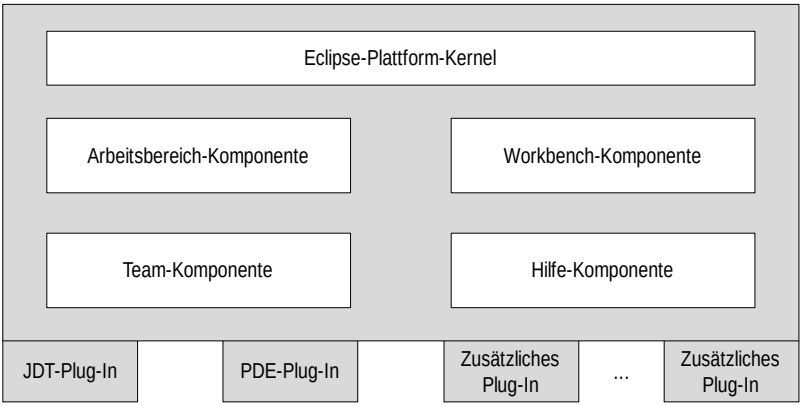


Bild 8-1: Eclipse-Architektur /60/

Der Plattform-Kernel bildet das Rückgrat der Eclipse-Plattform und sorgt dafür, dass die Plattform gestartet wird und alle notwendigen Plug-Ins geladen werden. Beim Start von Eclipse ist der Plattform-Kernel die Komponente, die zuerst ausgeführt wird. /60/

Die Arbeitsbereich-Komponente liefert das zentrale Ressourcenmanagement und bietet die Programmierschnittstelle für das Erzeugen und Verwalten von Projekten, Dateien und Ordnern. Die Workbench-Komponente bildet die graphische Benutzeroberfläche von Eclipse und definiert Erweiterungspunkte für das Hinzufügen von graphischen Elementen wie Editoren, Ansichten oder Menüeinträgen. Die Team- und Hilfe-Komponente

enthalten Funktionalitäten wie eine kontextsensitive Hilfe und den Zugriff auf Versionsverwaltungssysteme. Der Funktionsumfang von Eclipse wird durch die geladenen Plug-Ins bestimmt. /63/

8.1.2 Struktur der WieMod-Workbench

Die entwickelte WieMod-Workbench wurde modular aufgebaut, so dass sie zusätzlich zu den drei prototypisch integrierten Simulationswerkzeugen um beliebig viele weitere Simulationswerkzeuge erweitert werden kann. Sie umfasst daher insgesamt 57 eigens für WieMod entwickelte Plug-Ins und verwendet zusätzlich noch vorhandene Standard-Plug-Ins von Eclipse. In Bild 8-2 sind die wichtigsten Plug-Ins der WieMod-Workbench dargestellt.

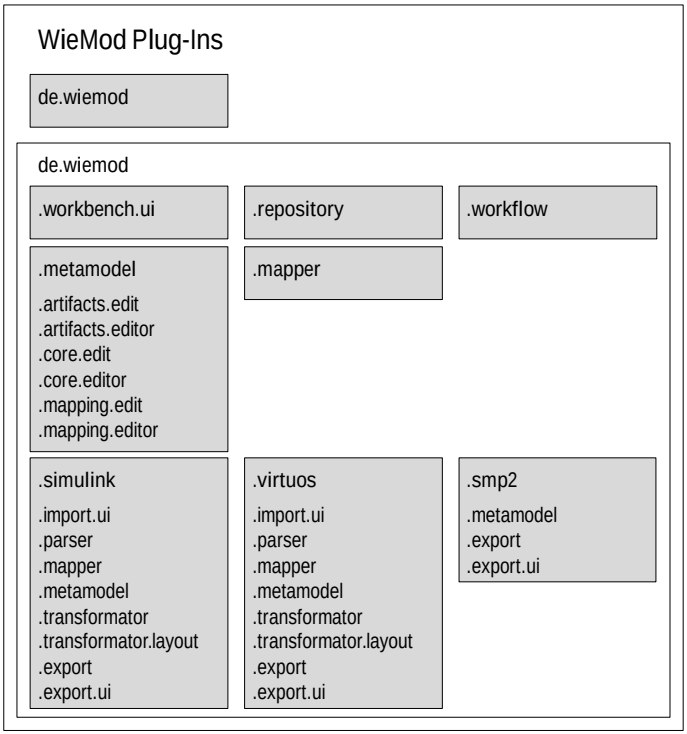


Bild 8-2: Übersicht der entwickelten WieMod Plug-Ins

Im Plug-In „de.wiemod.workbench.ui“ ist die Benutzeroberfläche (englisch: user interface (ui)) der WieMod-Workbench definiert. Hier sind die grundlegenden Eigenschaften der Benutzeroberfläche festgelegt. Inhalt und graphische Repräsentation von Menüs, Toolbars und Wizards werden hier eingestellt. In Kapitel 8.1.6 wird die Benutzeroberfläche der WieMod-Workbench mit ihren Funktionalitäten detailliert erläutert.

In „de.wiemod.repository“ befindet sich ein Repository, in dem alle verwendeten atomaren Komponenten (*Atomic Components*) der integrierten Simulationswerkzeuge mit den dazugehörigen Mappings hinterlegt sind. Darin können die atomaren Komponenten eines Simulationswerkzeugs, in der Implementierung als „Artefakte“ bezeichnet, und das Repository sowie die zugehörigen Mappings editiert werden. Das Repository kann beliebig um weitere Komponenten und Mappings erweitert werden. Die dazu benötigten Editoren und Menüs werden in diesem Plug-In definiert. Das Plug-In „de.wiemod.metamodel“ definiert die Elemente des WieMod Meta-Metamodells und beinhaltet die Implementierung für die Verwendung dieser Elemente innerhalb der WieMod-Workbench.

Die WieMod-Workbench benötigt verschiedene Workflows. Als solche werden die definierte Reihenfolge ausgeführter Befehle bezeichnet, die beispielsweise beim Import und Export von Simulationsmodellen ausgeführt werden. In „de.wiemod.mapper“ wird das Mapping der Modellelemente definiert. Dieses Plug-In ruft das für eine Transformation von einem Ausgangswerkzeug zu einem Zielwerkzeug benötigte Mapping, das im Repository der WieMod-Workbench manuell vorgenommen wurde, für den konkreten Anwendungsfall automatisch auf.

Für jedes in die WieMod-Workbench integrierte Simulationswerkzeug wurden mehrere Plug-Ins entwickelt, die die Wiederverwendung der werkzeugspezifischen Modelle ermöglichen. Für jedes Werkzeug wurde das werkzeugspezifische Metamodell implementiert. Für Matlab/Simulink und Virtuos ist die Struktur der Plug-Ins identisch. Deren Kombination ermöglicht den Import, das Parsen, das Mapping, die Transformation und den Export von werkzeugspezifischen Modellen. Die genaue Funktionsweise wird in Kapitel 8.1.5 erläutert.

Das Simulationswerkzeug SMP2 wurde nur für eine unidirektionale Nutzung in die WieMod-Workbench integriert, so dass aus der Meta-Metamodellebene zwar SMP2-Modelle exportiert, nicht jedoch SMP2-Modelle in die Meta-Metamodellebene impor-

tiert werden können. Da SMP2 im Gegensatz zu Matlab/Simulink und Virtuos nicht blockschaltbildbasiert ist, wird kein Mapping bezüglich einzelner Modellelemente vorgenommen. Die Transformation aus der Meta-Metamodellebene ist nicht wie bei den anderen Simulationswerkzeugen als eigenes Plug-In, sondern lediglich als einzelner Workflow umgesetzt.

8.1.3 Verwendete Frameworks

Als Framework (englisch für Rahmenstruktur) wird in der Softwaretechnik ein Programmiergerüst bezeichnet, das insbesondere im Rahmen der objektorientierten Softwareentwicklung als auch bei komponentenbasierten Entwicklungsansätzen Anwendung findet. Frameworks erleichtern die Programmierung komplexer Softwareanwendungen, indem sie die konzeptionelle Struktur der zu programmierenden Anwendung vorgeben und oftmals bereits fertige Codesegmente enthalten. /64/

Tabelle 8-1 bietet eine Übersicht der von der WieMod-Workbench verwendeten Frameworks zur Modellierung und zum Parsen. Diese werden im Folgenden kurz erläutert.

Modellierung	**Parsen**
EMF	DOM/DOM4J
MWE1/oAW4	
Xtend/Xpand/Check	

Tabelle 8-1: Übersicht der wichtigsten verwendeten Frameworks

Das **Eclipse Modeling Framework (EMF)** ist ein Modellierungsframework, das die Möglichkeit der Codegenerierung zur Erstellung von Applikationen, die auf einem strukturierten Datenmodell basieren, bietet. Aus einer Modellspezifikation in XMI (XML Metadata Interchange) können mit Hilfe von EMF Java-Klassen für das spezifische Modell erstellt werden. /65/

EMF-basierte Applikationen nutzen eine XML Schema Definition (XSD). Diese XSD stellt ein Modell und eine API (Application Programming Interface) zur Verfügung, mit

der die Komponenten des XML-Schemas bearbeitet werden können. Gleichzeitig wird der Zugriff auf die zugrunde liegende DOM (Document Object Model) Repräsentation des Schema-Dokuments gewährt. /66/

Eine **Modeling Workflow Engine (MWE)** ist ein deklarativer, konfigurierbarer Code-Generator, der eine einfache XML-basierte Konfigurationssprache zur Verfügung stellt. Mit dieser kann die Verknüpfung einer Vielzahl verschiedener Workflows beschrieben werden. Eine solche Verknüpfung besteht aus mehreren Workflowkomponenten, die sequenziell in einer Java Virtual Machine (JVM) ausgeführt werden. Unter dem Namen **MWE 1** wurde die **openArchitectureWare 4 (oAW 4)** zum Eclipse Modeling Framework hinzugefügt. In der WieMod-Workbench wird MWE 1 zur Erstellung von Workflows zur Modelltransformation verwendet.

Das **Xpand Framework** stellt textuelle Sprachen zur Verfügung, die den modellbasierten Softwareentwicklungsprozess, zum Beispiel bei der Codegenerierung oder Modelltransformationen, unterstützen. Die enthaltenen Sprachen Check, Xtend und Xpand basieren auf dem gleichen Satz von logischen Ausdrücken und dem gleichen Typsystem. Deshalb können sie auf die gleichen Modelle, Metamodelle und Meta-Metamodelle kombiniert angewandt werden. Die Syntax ist bei allen drei Sprachen gleich, was die Softwareentwicklung sehr vereinfacht. /67/ Sämtliche Transformationen in das WieMod Meta-Metamodell und aus dem WieMod Meta-Metamodell hinaus werden unter Verwendung des Xpand Frameworks durchgeführt.

Das **Document Object Model (DOM)** ist eine plattformübergreifende und programmiersprachenunabhängige Konvention zur Interaktion mit Objekten in HTML-, XHTML- und XML-Dokumenten. Die Elemente in DOM können mit der jeweils verwendeten Programmiersprache adressiert und bearbeitet werden. DOM4J ist die entsprechende Open Source Bibliothek für die Java-Plattform. Sie verwendet das Java Collections Framework und unterstützt in vollem Umfang DOM, SAX und JAXP, welches Plattformen zum Parsen und Validieren von XML-Dokumenten sind. /68/ DOM wurde in der WieMod-Workbench zur Realisierung sämtlicher Parser eingesetzt.

8.1.4 Metamodellierung in Eclipse

Zur einmalig notwendigen Modellierung der werkzeugspezifischen Metamodelle und des Meta-Metamodells in Eclipse wurde das Eclipse Modeling Framework verwendet. Dieses basiert selber auf zwei Metamodellen, dem Ecore- und dem Genmodel-Metamodell. Das Ecore-Metamodell beinhaltet Informationen zu den definierten Klassen. Das Genmodel-Metamodell enthält zusätzliche Informationen für die Codegenerierung. Das Ecore-Metamodell erlaubt die Definition der Elemente EClass, EAttribute, EReference und EDataType. EClass repräsentiert eine Klasse mit Attributen und Referenzen. Dabei bildet die Klasse EAttribute ein Attribut mit einem Namen und einem Typ. Die Klasse „EReference“ stellt die Beziehung zwischen zwei Klassen dar. In der Klasse „EDataType„ wird schließlich der Typ eines Attributs, wie zum Beispiel float oder int festgelegt. /69/

Ein Ecore-Metamodell wird in einer Baumstruktur dargestellt. Diese verfügt immer über ein Wurzelelement, welches das gesamte Modell repräsentiert. Dieses enthält Zweige, die die Packages des entsprechenden Plug-Ins repräsentieren. Die Zweige der Packages stellen die einzelnen Klassen dar. Deren Zweige repräsentieren wiederum die Attribute der Klassen.

Das WieMod Meta-Metamodell ist im Plug-in „de.wiemod.metamodel“ in insgesamt vier Ecore-Metamodellen modelliert (Bild 8-3).

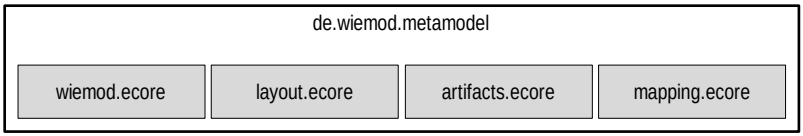


Bild 8-3: Ecore-Metamodelle zur Beschreibung des WieMod Meta-Metamodells in Eclipse

In „wiemod.ecore“ wird das Semantik-Modell des in Kapitel 6 entwickelten WieMod Meta-Metamodells abgebildet. Das Ecore-Metamodell „layout.ecore“ beschreibt das Layout-Modell des WieMod Meta-Metamodells. Zusätzlich werden in „artifacts.ecore“ noch die Artefakte, die mit Hilfe der WieMod-Workbench aus Modellen der verschiede-

nen Simulationswerkzeuge extrahiert werden können, im Metamodell definiert. In „mapping.ecore“ wird das Block-Mapping beschrieben.

In Bild 8-4 ist anhand des Modells „wiemod.ecore“ exemplarisch der Aufbau eines E-core-Metamodells mit Wurzelelement, Package, Klassen und Attributen zu sehen.

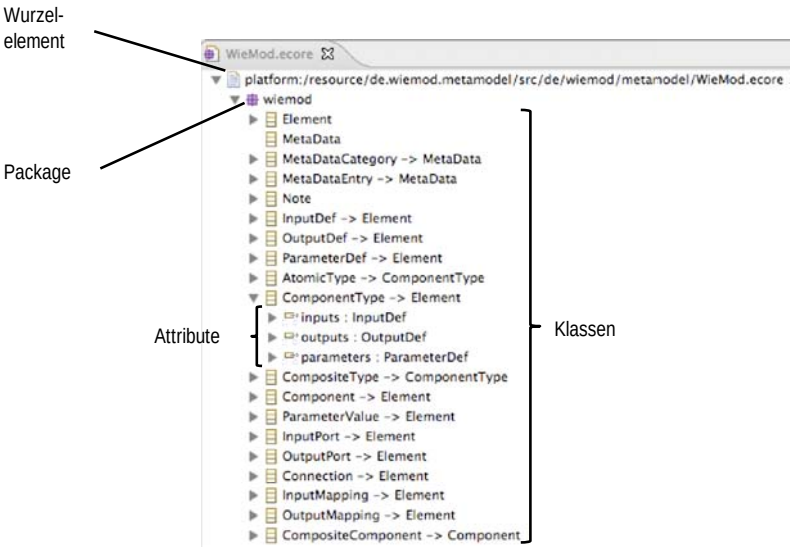


Bild 8-4: Modellierung des Semantikmodells des WieMod Meta-Metamodells als E-core-Metamodell in Eclipse

Das Matlab/Simulink-Metamodell ist in Eclipse ebenfalls in Form eines Ecore-Metamodells abgebildet, wie in Bild 8-5 zu sehen ist. Da Matlab/Simulink-Modelle blockschaltbildbasiert sind, enthält das Matlab/Simulink-Metamodell neben semantischen, zusätzliche graphische Informationen zum Layout des Blockschaltbilds.

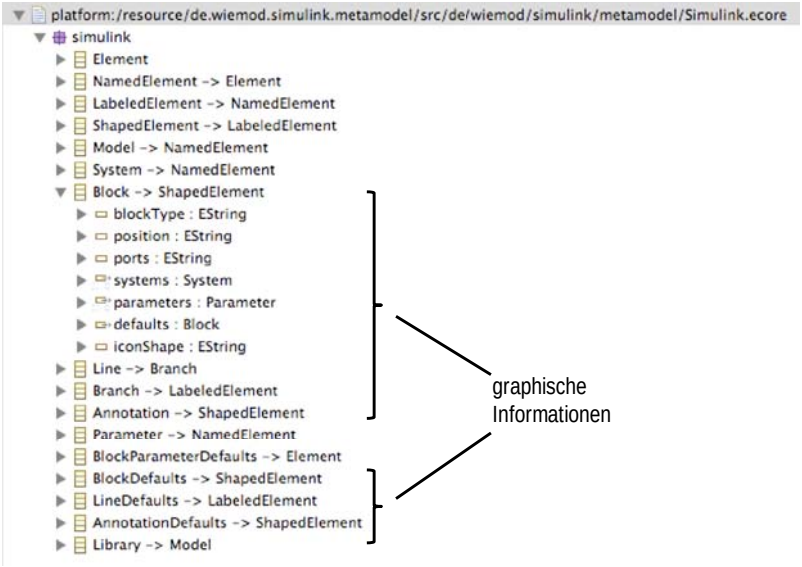


Bild 8-5: Ecore-Schema des Matlab/Simulink-Metamodells

Das Virtuos-Metamodell wird im Gegensatz zum Matlab/Simulink-Metamodell nicht in Form eines Ecore-Metamodells abgebildet, sondern ist entsprechend dem Aufbau von Virtuos-Modelldateien strukturiert und basiert deshalb wie die Virtuos-Modelldateien selbst auf XML. In Bild 8-6 ist ein Auszug aus dem XML-Schema des Virtuos-Metamodells in Eclipse zu sehen. Da Virtuos-Modelle in einem Blockschaltbild modelliert werden, enthält das Virtuos-Metamodell neben semantischen Informationen zusätzlich graphische Informationen, die das Layout des Blockschaltbilds beschreiben.

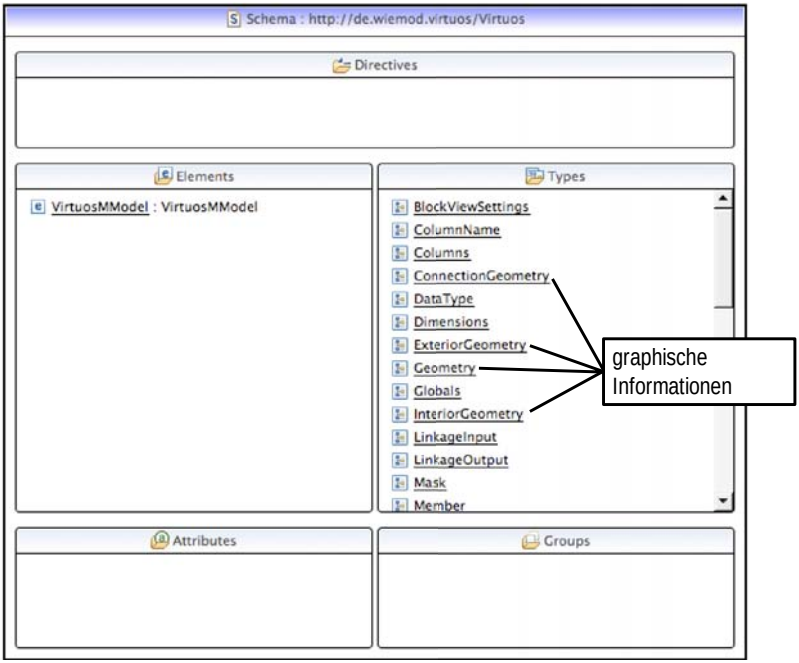


Bild 8-6: XML-Schema des Virtuos-Metamodells

Das SMP2-Metamodell basiert als Abstraktion der XML-basierten SMP2-Modelldateien ebenfalls auf XML. Allerdings besteht das SMP2-Metamodell nur aus einem Semantikmodell, da SMP2-Modelle über keine graphische Repräsentation verfügen. Aus diesem Grund enthält das SMP2-Metamodell ausschließlich ein Semantikmodell und verfügt über kein zusätzliches Layoutmodell.

8.1.5 Realisierung von Mapping und Modelltransformationen

Die in Kapitel 7.2 erarbeiteten Mapping-Strategien und die Methoden zur Modelltransformation wurden in der WieMod-Workbench prototypisch umgesetzt. Sowohl Mapping als auch Transformation erfolgen in zwei Schritten. Im ersten Schritt erfolgt ein Mapping der Modellelemente eines Modells aus einem Ausgangswerkzeug auf die entsprechenden Modellelemente im Meta-Metamodell. Die Transformation erfolgt als Anwendung des Mappings ebenfalls zunächst von der Modelldatei des Ausgangswerkzeugs auf

eine Modelldatei im Meta-Metamodell. Im zweiten Schritt werden sowohl das Mapping als auch die Transformation der Modellelemente aus dem Meta-Metamodell in Modellelemente für das gewünschte Zielwerkzeug durchgeführt. Diese beiden Schritte werden in diesem Unterkapitel detailliert erläutert.

In Bild 8-7 ist die detaillierte Prozesskette des ersten Schritts von Mapping und Transformation, nämlich vom Ausgangswerkzeug zum Meta-Metamodell, dargestellt. Die für die einzelnen Prozessschritte benötigten Plug-Ins sind über dem jeweiligen Schritt aufgelistet.

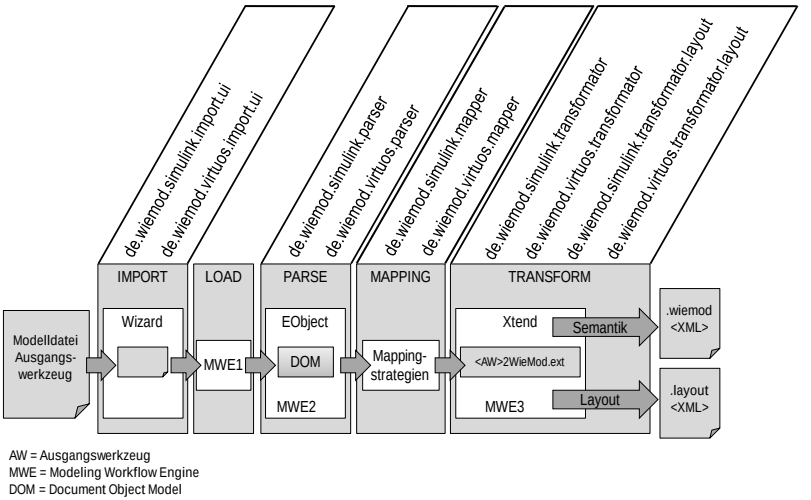


Bild 8-7: Detaillierte Prozesskette der Modelltransformation vom Ausgangswerkzeug zum Meta-Metamodell

Dabei wird zunächst eine Modelldatei eines Ausgangswerkzeugs mit Hilfe eines Wizards, einer Benutzeroberfläche mit verschiedenen Dialogfenstern zur Unterstützung bei der Dateneingabe, importiert. In der prototypischen Realisierung wurden die Simulationswerkzeuge Matlab/Simulink und Virtuos als Ausgangswerkzeuge in die WieMod-Workbench integriert. Dementsprechend existiert für jedes dieser Simulationswerkzeuge ein Plug-In, in dem die Import-Wizards implementiert sind. Die verschiedenen implementierten Wizards werden in Kapitel 8.1.6 vorgestellt. Alle Java-Klassen, die die Fens-

ter und Seiten des Wizards beschreiben, sind in den werkzeugspezifischen Plug-Ins („<simulationswerkzeug>.import.ui“) gespeichert. In jedem dieser Plug-Ins befinden sich wiederum zwei Packages. Eines definiert die Reihenfolge der ausgeführten Befehle, die den eigentlichen Wizard öffnen (Dateiendung „.actions“), das andere den Wizard selbst (Dateiendung „.wizards“). Das importierte Modell ist entsprechend des jeweiligen werkzeugspezifischen Metamodells beschrieben und ist jeweils im Plug-In „<simulationswerkzeug>.metamodel“ zu finden. Der Import-Wizard für Matlab/Simulink-Modelle verfügt über eine zusätzliche Seite, die dem Nutzer die Möglichkeit bietet, einzelne atomare Komponenten (so genannte Artefakte) des Modells auszuwählen und diese als Typen zu importieren. Dazu wird das Modell in seine atomaren Komponenten zerlegt, die in einer Baumstruktur dargestellt werden. Dieser Schritt ist optional.

Damit die Modelldatei weiter verarbeitet werden kann, muss diese geladen und in einem temporären Verzeichnis gespeichert werden. Dies erfolgt mit Hilfe der ersten Modeling Workflow Engine (MWE1). Für das Simulationswerkzeug Matlab/Simulink wurde die Java-Klasse „MdlReader.java“ erstellt. Da es sich bei Simulink-Modelldateien um ein proprietäres Dateiformat handelt, musste dafür eine eigene Klasse zum Einlesen entwickelt werden. Virtuos-Modelldateien sind XML-basiert, so dass diese mit einem einfachen XML-Reader eingelesen werden können. Dieser ist in der Java-Klasse „EcfReader.java“ implementiert.

Im nächsten Schritt werden die in der Modelldatei enthaltenen Modellinformationen analysiert. Dies erfolgt jeweils mit Hilfe eines werkzeugspezifischen Parsers, der im Package „<simulationswerkzeug>.parser“ implementiert ist. Die Java-Klasse zum Parsen von Matlab/Simulink-Modelldateien heißt „SimulinkParser.java“, die entsprechende Klasse zum Parsen von Virtuos-Modelldateien heißt „VirtuosParser.java“. Beim Parsen werden die Dateien in EObjects umgewandelt und im lokalen Speicher abgelegt. Die entsprechenden Klassen dazu heißen „SimulinkTransformator.java“ bzw. „VirtuosTransformator.java“. Dies erfolgt mit Hilfe der zweiten Modeling Workflow Engine (MWE2).

Im nächsten Schritt erfolgt das Mapping der Modellelemente. Dies wurde mit Hilfe der Java-Klassen „SimulinkTypeMapper.java“ und „VirtuosTypeMapper.java“ in den Packages „de.wiemod.<simulationswerkzeug>.mapper“ implementiert. Hier werden die manuell in der WieMod-Workbench definierten Mappingsstrategien aufgerufen. Die manuelle Definition der Mappings wird in Kapitel 8.1.6 beschrieben.

Im nächsten Schritt erfolgt dann die eigentliche Modelltransformation, bei der die jeweiligen Mappings angewendet werden. Die Regeln, nach denen die Transformation der Semantik und des Layouts durchgeführt werden, sind in zwei Dateien definiert. Diese haben jeweils den Namen „<Ausgangswerkzeug>2WieMod.ext“ und sind in den Plug-Ins „<simulationswerkzeug>.transformator“ und „<simulationswerkzeug>.transformator.layout“ hinterlegt. Jedes transformierte Modellelement wird in einen in Extend definierten Speicherbereich, einen so genannten Memory-Slot, geschrieben.

Nachdem die Transformationen von Semantik und Layout der einzelnen Modelldateien ins Meta-Metamodell erfolgt ist, werden diese in neue Modelldateien geschrieben. Die dafür zuständige Java-Klasse „Writer.java“ wurde im Plug-In „de.wiemod.common“ implementiert. Sie fügt die transformierten Modellelemente aus den Memory-Slots zu einem neuen Gesamtmodell zusammen. Der Dateityp und die Dateiendung werden durch die dritte Modeling Workflow Engine (MWE3) generiert. Das Resultat dieser Prozessschritte sind zwei Dateien, davon eine mit der Endung „.wiemod“, in der die Semantik des transformierten Modells gespeichert ist und eine mit der Endung „.layout“, in der die graphische Repräsentation des Modells hinterlegt ist.

In den folgenden Bildern (Bild 8-8 und Bild 8-9) sind Codeausschnitte aus diesen beiden Dateien zu sehen. Als Beispielmodell wurde wieder das Modell „AddConstants“ gewählt. Bild 8-8 zeigt zunächst einen Codeausschnitt aus der Meta-Metamodelldatei „AddConstants.wiemod“, in dem die Semantik des Modells „AddConstants.mdl“ aus dem Simulationswerkzeug Matlab/Simulink in das Meta-Metamodell transformiert wurde.

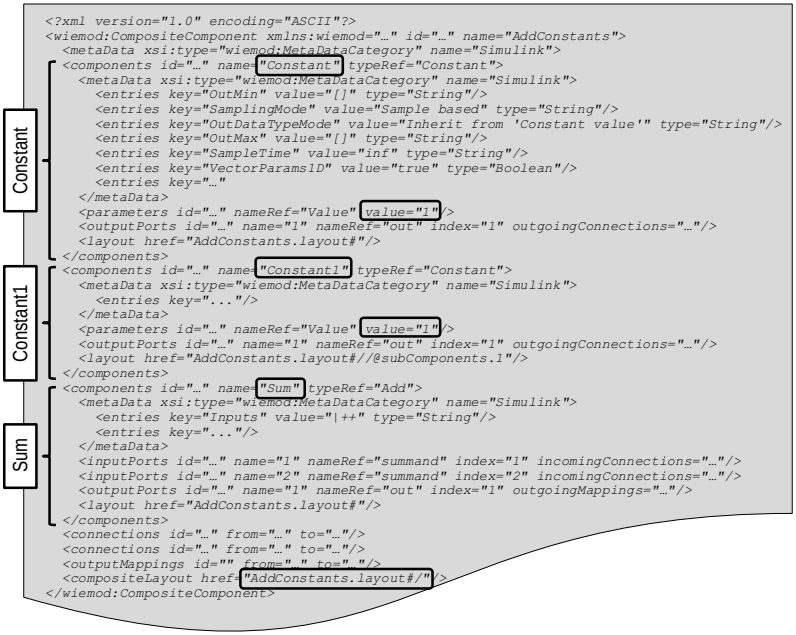


Bild 8-8: Codeausschnitt aus der Meta-Metamodelldatei „AddConstants.wiemod“

Die Meta-Metamodelldatei ist XML-basiert. In ihrem Header werden zunächst allgemeine Angaben zu XML-Version, Modellnamen etc. gemacht. Im darauffolgenden Tag <metadata> wird das Ausgangswerkzeug der Transformation, in diesem Fall Matlab/Simulink, benannt. Die nächsten drei Tags <components> beinhalten jeweils einen der drei im Modell enthaltenen Blöcke „Constant“, „Constant1“ und „Sum“. Diese werden im Bild durch Markierungen hervorgehoben. Zu jedem Block werden in dem Tag <metaDataEntries> allgemeine Angaben zu Rechenschrittweite und Datentypen gemacht. Darauf folgen Angaben zu den Eingangs- und Ausgangsports jedes Blocks. Am Ende der Beschreibung aller im Modell enthaltenen Blöcke wird mit dem Tag <layout> auf die zugehörige Datei mit der Beschreibung der graphischen Repräsentation des Modells verwiesen, in diesem Fall „AddConstants.layout“.

In Bild 8-9 ist ein Codeausschnitt aus der Meta-Metamodelldatei „AddConstants.layout“ zu sehen. Die Datei beginnt genau wie die Datei „AddConstants.wiemod“ mit einem

Header, der allgemeine Informationen enthält. Diese Datei ist zwar prinzipiell menschenlesbar, allerdings ist sie schwieriger zu interpretieren, da statt der Namen der Blöcke lediglich die entsprechenden Identifikationsnummern aus der Datei „AddConstants.wiemod“ mit einem Verweis auf diese Datei aufgeführt sind. Die Datei „AddConstants.layout“ enthält Zusatzinformationen zu der graphischen Repräsentation des Modells im Blockschaltbild. Sie besteht überwiegend aus Koordinaten, die die Position der Blöcke im Tag <position> sowie den Verlauf der Signalverbindungen zwischen den Blöcken im Tag <points> beschreiben.

```
<?xml version="1.0" encoding="ASCII"?>
<wiemod.layout:CompositeComponentLayout xmi:version="2.0"
        xmlns:wiemod.layout="…" name="AddConstants">
  <position>
    <downLeft x="10" y="10"/>
    <upperRight x="110" y="60"/>
  </position>
  <component href="AddConstants.wiemod#/"/>
  <outputPorts id="08e89e63-2588-4f51-b59c-1ed484daf2ec">
    <outputPort href="AddConstants.wiemod#//@outputPorts.0"/>
    <position x="110" y="35"/>
    <innerPosition x="425" y="175"/>
    <innerBlockPosition>
      <downLeft x="425" y="168"/>
      <upperRight x="455" y="182"/>
    </innerBlockPosition>
  </outputPorts>
  <innerPosition>
    <downLeft x="214" y="78"/>
    <upperRight x="1278" y="974"/>
  </innerPosition>
  <connectionsLayout fromPort="//@subComponents.0/@outputPorts.0">
    <connections href="AddConstants.wiemod#//@connections.0"/>
    <to id="761ae113-5e10-4172-a7de-c407d51faec1">
      <connections href="AddConstants.wiemod#//@connections.0"/>
      <to xsi:type="…" toPort="//@subComponents.2/@inputPorts.0"/>
      <points x="285" y="135"/>
      <points x="290" y="135"/>
      <points x="290" y="175"/>
      <points x="300" y="155"/>
    </to>
  </connectionsLayout>
  <compositeComponent href="AddConstants.wiemod#/"/>
</wiemod.layout:CompositeComponentLayout>
```

Bild 8-9: Codeausschnitt aus der Meta-Metamodelldatei „AddConstants.layout“

Diese beiden Dateien bilden einerseits das Ergebnis der ersten Transformation, der Transformation vom Ausgangswerkzeug zum Meta-Metamodell. Gleichzeitig bilden sie die Eingangsdateien für den zweiten Schritt, der Mapping und Transformation vom Meta-Metamodell zum gewünschten Zielwerkzeug beinhaltet. In der prototypischen Reali-

sierung wurden als Zielwerkzeuge die Simulationswerkzeuge Matlab/Simulink, Virtuos und SMP2 in die WieMod-Workbench integriert. In Bild 8-10 ist die detaillierte Prozesskette vom Meta-Metamodell zum Zielwerkzeug dargestellt.

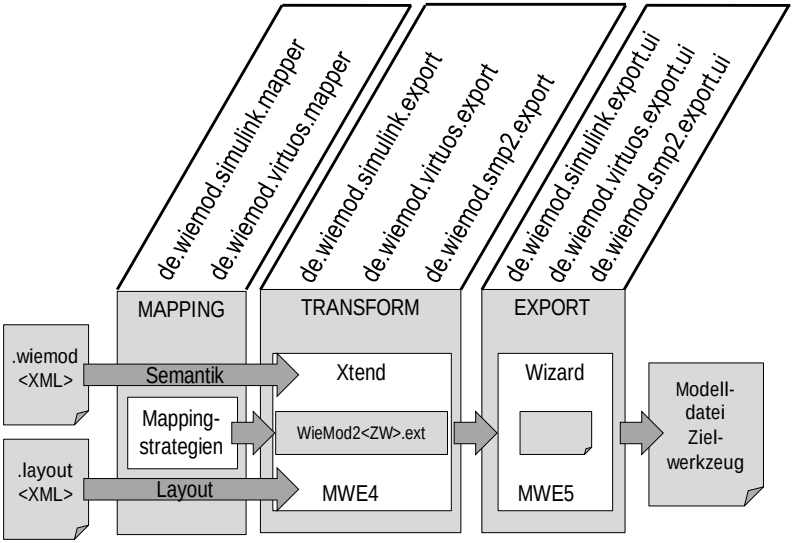


Bild 8-10: Detaillierte Prozesskette der Modelltransformation vom Meta-Metamodell zum Zielwerkzeug

Der erste Schritt der Modelltransformation vom Meta-Metamodell zum Zielwerkzeug besteht wieder aus einem Mapping der Modellelemente. In diesem Fall werden die Modellelemente des Meta-Metamodells auf die entsprechenden Modellelemente des Zielwerkzeugs abgebildet. Dies erfolgt wieder mit Hilfe der Java-Klassen „SimulinkTypeMapper.java“ bzw. „VirtuosTypeMapper.java“ aus den Packages „de.wiemod.<simulationswerkzeug>.mapper“, indem die manuell in der WieMod-Workbench definierten Mapping-Strategien aufgerufen werden.

Im zweiten Schritt erfolgt die eigentliche Modelltransformation. Dabei werden die zuvor durchgeführten Mappings angewandt, indem die Modellelemente des Meta-Metamodells durch die entsprechenden Modellelemente des Zielwerkzeugs ersetzt werden. Dieser Vorgang wird durch die insgesamt vierte Modeling Workflow Engine (MWE4) durchgeführt. Die Regeln, nach denen die Transformation der Semantik und des Layouts aus den beiden Dateien durchgeführt werden, sind in jeweils einer Datei definiert. Diese hat den Namen „WieMod2<Zielwerkzeug>.ext“ und ist im jeweiligen Plug-In „de.wiemod.<simulationswerkzeug>.export“ hinterlegt. Jedes transformierte Modellelement wird wieder in einen Memory-Slot geschrieben.

Nachdem die Transformationen von Semantik und Layout der einzelnen Meta-Metamodelldateien in das gewünschte Zielwerkzeug erfolgt sind, werden diese neuen Modelldateien exportiert. Dies erfolgt mit Hilfe eines Wizards, der sich für jedes Zielwerkzeug im entsprechenden Plug-In „de.wiemod.<simulationswerkzeug>.export.ui“ befindet. Wie die Import-Wizards enthalten die Export-Wizards zwei Packages. Eines definiert die Reihenfolge der ausgeführten Befehle, die den eigentlichen Wizard öffnen (Dateiendung „.actions“), das andere den Wizard selbst (Dateiendung „.wizards“). Zum Schluss werden die transformierten Modellelemente aus den Memory-Slots zu einem neuen Gesamtmodell zusammengefügt. Der Dateityp und die Dateiendung für das Zielwerkzeug werden durch die fünfte Modeling Workflow Engine (MWE5) generiert. Das Resultat dieser Prozessschritte ist eine Modelldatei für das gewünschte Zielwerkzeug, in der sowohl die Semantik des transformierten Modells als auch die graphische Repräsentation des Modells gespeichert sind.

8.1.6 Benutzeroberfläche der WieMod-Workbench

Die Benutzeroberfläche der WieMod-Workbench wird in diesem Unterkapitel vorgestellt. Über „wiemod.product“ im Plug-In „de.wiemod“ kann die WieMod-Workbench aus Eclipse gestartet werden. Darüber hinaus kann sie als Standalone-Anwendung ausgeführt werden. Im Ordner „Release“ in der Entwicklungsumgebung werden dazu Installationsdateien für die Betriebssysteme Windows und Linux bzw. MacOS automatisch generiert.

Die WieMod-Workbench basiert wie die Entwicklungsumgebung ebenfalls auf Eclipse. In Bild 8-11 ist der Aufbau des Hauptfensters der WieMod-Workbench dargestellt.

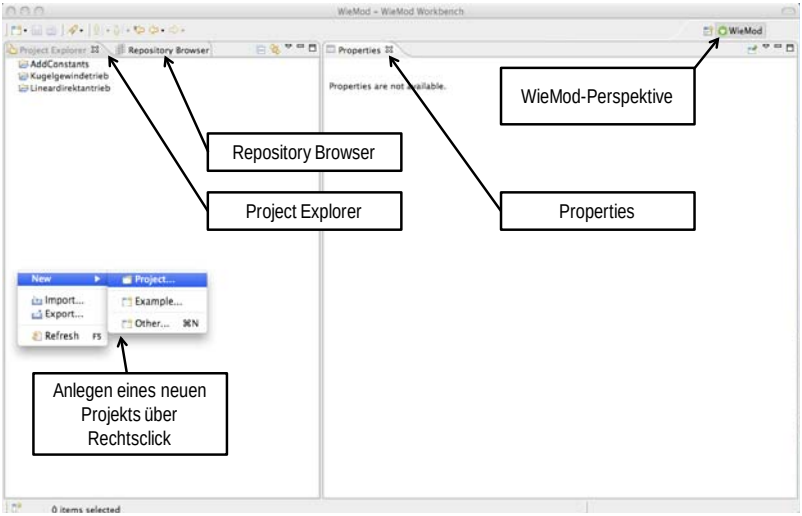


Bild 8-11: Project Explorer im Hauptfenster der WieMod-Workbench

Die Menüleisten sind wie bei jeder Eclipse-Anwendung standardmäßig aufgebaut. Bei der eigens für WieMod erstellten WieMod-Perspektive ist das Hauptfenster in zwei Bereiche unterteilt. Auf der linken Seite befinden sich zwei Tabs, mit denen wahlweise der Project Explorer oder der Repository Browser eingeblendet werden. Auf der rechten Seite befindet sich ein Tab, in dem die Eigenschaften eines angewählten Elements angezeigt werden.

Im Project Explorer können Projektordner für jedes zu transformierende Simulationsmodell angelegt werden. Über einen Rechtsklick erscheint ein Dialog zum Anlegen eines neuen Projektordners.

In einen solchen Ordner können nun Modelle aus den Simulationswerkzeugen Matlab/Simulink und Virtuos importiert werden. Über einen Rechtsklick auf den gewünschten Ordner kann über Anwählen der Schaltfläche „Import...“ der Import-Wizard geöffnet werden (Bild 8-12).

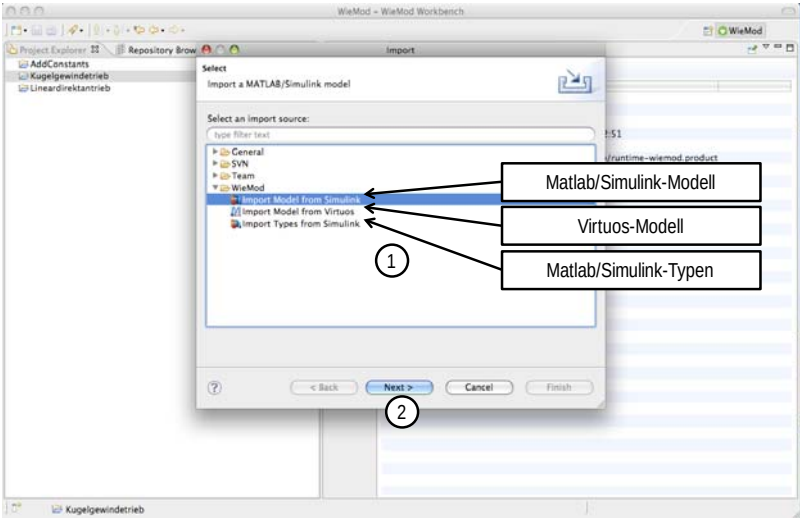


Bild 8-12: Import-Wizard

Dort wird zunächst ausgewählt, aus welchem Simulationswerkzeug das zu importierende Modell stammt. Durch einen Klick auf die Schaltfläche „Next“ öffnet sich ein Dialog, in dem der Pfad zu dem gewünschten Modell sowie der Speicherort für die ins Meta-Metamodell transformierten Dateien angegeben werden. Diese Funktionalität wird aus dem jeweiligen Plug-In „de.wiemod.<ausgangswerkzeug>.import.ui“ aufgerufen. Durch einen Klick auf die Schaltfläche „Finish“ werden die entsprechenden Workflow angestoßen, die die Modelldatei laden (MWE1) und parsen (MWE2). Im Hintergrund werden die entsprechenden Mapping-Strategien aufgerufen, deren Anwendung in der anschließenden Transformation durch einen weiteren Workflow (MWE3) ausgeführt werden. Auf diese Weise wird eine Modelldatei aus dem gewählten Ausgangswerkzeug in zwei Meta-Metamodelldateien transformiert. Die Datei mit der Endung „.wiemod“ enthält die semantischen Modellinformationen, die Datei mit der Endung „.layout“ die graphischen Modellinformationen.

In Bild 8-13 ist das Ergebnis der beschriebenen Prozessschritte zu sehen. Die entsprechenden Dateien für Semantik- und Layout-Modell der Matlab/Simulink-Modelldatei „Kugelgewindetrieb.mdl“ wurden dabei von der WieMod-Workbench generiert. Im

Fenster „Properties“ auf der rechten Seite werden grundsätzliche Informationen des Projektordners „Kugelgewindetrieb“ angezeigt.

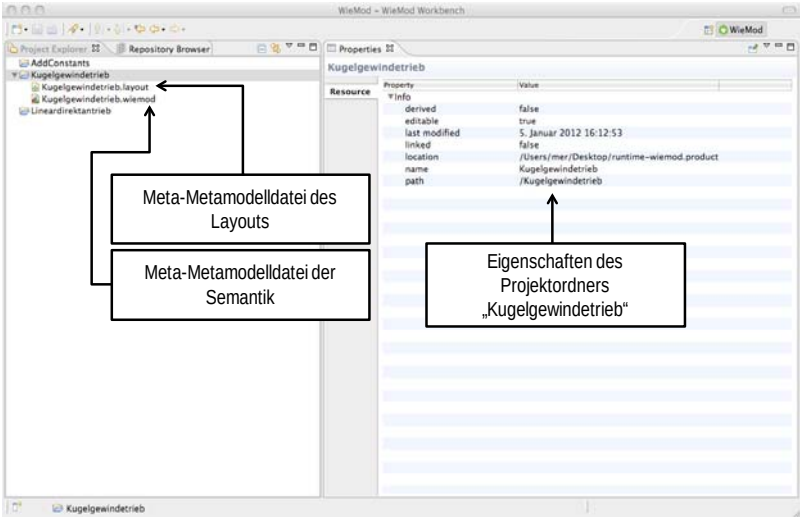


Bild 8-13: Ergebnis des Imports einer Modelldatei

Durch einen Rechtsklick auf eine der beiden neu generierten Meta-Metamodelldateien wird ein Kontextmenü zur Modelltransformation geöffnet. Ob die Meta-Metamodelldatei des Layouts oder der Semantik angewählt wird, spielt keine Rolle, da beide Dateien aufeinander verweisen und dadurch automatisch beide Dateien berücksichtigt werden. Das Menü ist in Bild 8-14 zu sehen. Da in der prototypischen Realisierung die Simulationswerkzeuge Matlab/Simulink, Virtuos und SMP2 als Zielwerkzeuge implementiert wurden, stehen diese drei Simulationswerkzeuge im Export-Wizard zur Verfügung.

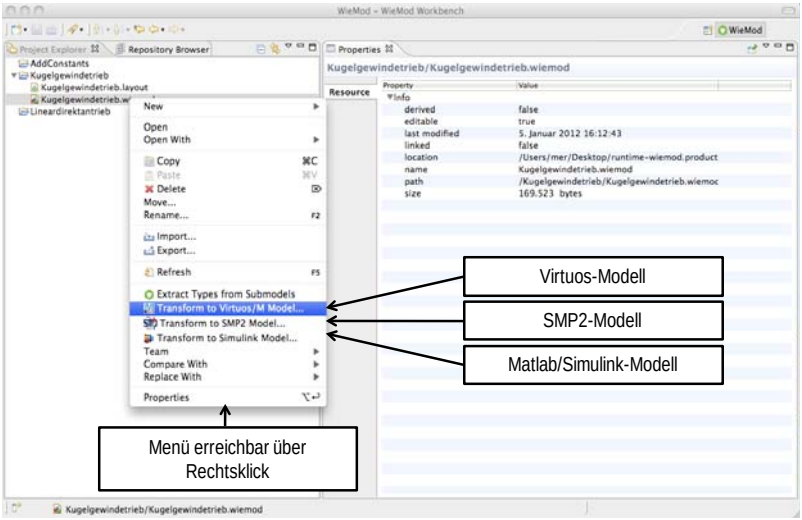


Bild 8-14: Export-Wizard

Nach dem Auswählen eines der drei verfügbaren Zielwerkzeuge öffnet sich erneut ein Dialogfenster. Darin ist die zuvor erfolgte Dateiauswahl bereits markiert. Von dem Nutzer muss noch der Pfad für das transformierte Modell angegeben werden. Durch einen Klick auf die Schaltfläche „Finish“ wird der Workflow für die gesamte Prozesskette zur Modelltransformation vom Meta-Metamodell zum Zielwerkzeug angestoßen. Dabei werden zunächst die entsprechenden Mapping-Strategien aus den Plug-Ins „de.wiemod.<zielwerkzeug>.mapper“ aufgerufen. Die jeweilige Modeling Workflow Engine aus dem Plug-In de.wiemod.<zielwerkzeug>.export (MWE4) führt die Anwendung der Mapping-Strategien aus. Schließlich wird im letzten Workflow (MWE5) die Datei für das gewählte Zielwerkzeug geschrieben.

Über den Tab „Repository Browser“ im Hauptfenster der WieMod-Workbench kann zum Repository Browser gewechselt werden (Bild 8-15). Dort befindet sich eine Liste aller verfügbaren *Atomic Components*. Durch einen Rechtsklick auf eine *Atomic Component* können die Input-, Output- und Parameterdefinitionen jeder Komponente angelegt werden. Jedes Element der Liste kann aufgeklappt werden, so dass die angelegten Mappings angezeigt werden. Durch einen Doppelklick auf eines dieser Mappings werden auf der rechten Seite zwei neue Fenster geöffnet. Im oberen Bereich kann durch ei-

nen Rechtsklick ein neues Mapping angelegt werden, das im unteren Bereich editiert werden kann. Unter „Source Platform“ wird der Name des Ausgangswerkzeugs angegeben, unter „Target Platform“ der Name des Zielwerkzeugs. Mit Source wird der Name des Modellelements im Ausgangswerkzeug, mit Target der Name des Modellelements im Zielwerkzeug bezeichnet. In diesem Fenster kann die WieMod-Workbench um beliebig viele weitere *Atomic Components* und Mappings verschiedener Simulationswerkzeuge erweitert werden.

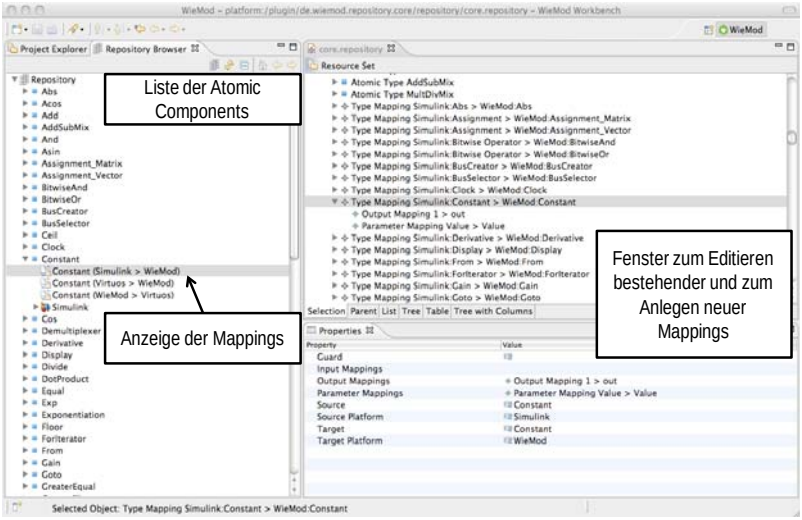


Bild 8-15: Repository Browser im Hauptfenster der WieMod-Workbench

Anlegen eines komplexen Mappings am Beispiel „AddSubMix“

In diesem Abschnitt wird am Beispiel eines gemischten Additions-/Subtraktionsblocks in Matlab/Simulink (vgl. Kapitel 7.1, Abschnitt „Fehlendes Modellelement“) gezeigt, wie das manuelle Anlegen einer neuen *Atomic Component* und eines neuen komplexen Mappings konkret durchgeführt wird. Zunächst wird dabei im Repository Browser eine neue *Atomic Component* mit dem Namen „AddSubMix“ hinzugefügt. Anschließend

werden der Eingang „in“, der Ausgang „out“ und der Parameter „InputSignums“ angelegt (Bild 8-16).

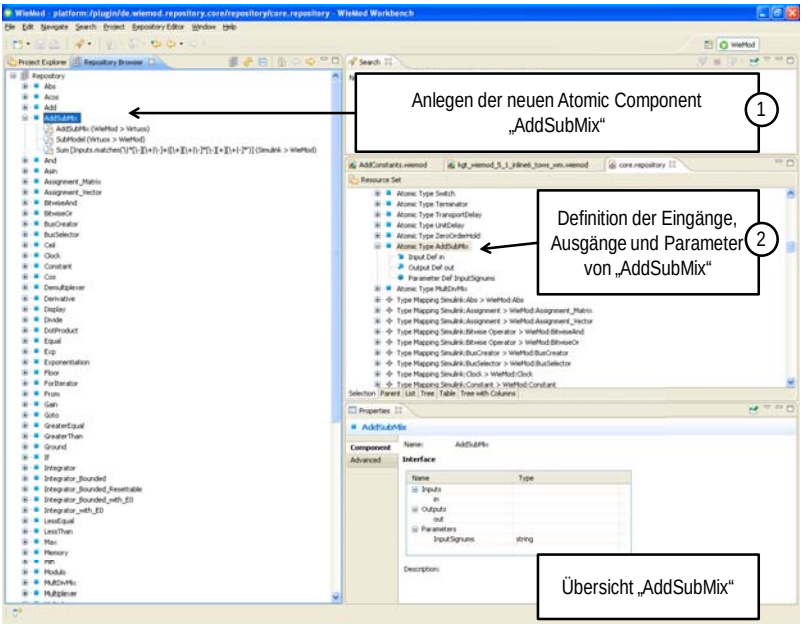


Bild 8-16: Anlegen einer neuen *Atomic Component*

Im Anschluss werden die Mapping-Strategien definiert. Im ersten Schritt wird dabei das Mapping von dem Modellelement „Sum“ im Ausgangswerkzeug „Matlab/Simulink“ (E1) auf die neu angelegte *Atomic Component* „AddSubMix“ im WieMod Meta-Metamodell (E2) definiert. Im Input-Mapping wird der Eingang „1“ des Modellelements „Sum“ in Matlab/Simulink auf den Eingang „in“ im WieMod Meta-Metamodell abgebildet. Analog erfolgt das Output-Mapping von dem Ausgang „1“ in Matlab/Simulink auf „out“ im WieMod Meta-Metamodell. Da im WieMod Meta-Metamodell nicht die Möglichkeit besteht, Eingänge mit Vorzeichen zu versehen, wie es für die Definition des Modellelements des gemischten Additions-/Subtraktionsblocks notwendig ist, wird diese fehlende Funktionalität über die Parameter nachgebildet. Im konkreten Beispiel wird die Folge der Vorzeichen von „Sum“ auf den neu definierten Parameter „InputSignums“ abgebildet. Um sicherzustellen, dass diese komplexe Mapping-Strategie nicht automatisch

auch für einfache Additions- und Subtraktionsmodellelemente mit nur zwei Eingängen an-gewandt wird, wird über einen so genannten „Guard“ die Bedingung festgelegt, unter der die Mapping-Strategie zum Einsatz kommt. Dies ist der Fall, wenn das Modellelement „Sum“ in Matlab/Simulink mindestens über drei Eingänge verfügt und diese gemischte Vorzeichen haben (Bild 8-17).

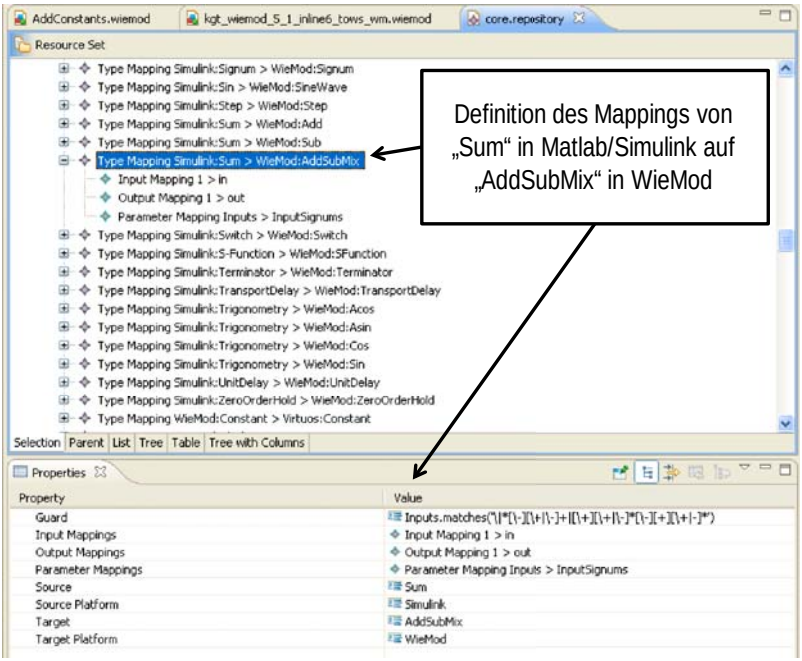


Bild 8-17: Definition des Mappings vom Ausgangswerkzeug zum Meta-Metamodell

Im zweiten Schritt wird das Mapping der *Atomic Component* „AddSubMix“ im WieMod Meta-Metamodell (E2) auf das Modellelement „SubModel“ im Zielwerkzeug Virtuos (E3) definiert. Dabei wird der Eingang „in“ von „AddSubMix“ im WieMod Meta-Metamodell auf den Ein-gang „inp1“ des Modellelements „SubModel“ in Virtuos abgebildet. Analog erfolgt ein Mapping des Ausgangs „out“ von „AddSubMix“ auf den Ausgang „out“ des Modellelements „SubModel“ in Virtuos. Der Parameter „InputSignums“ in Matlab/Simulink wird auf einen neu definierten Parameter „GenericParam“ in Virtuos abgebildet. Für diesen muss eine neue komplexe Mapping-Strategie festgelegt werden.

Dazu wird eine so genannte „Meta Data Entry“ zu dem Mapping hinzugefügt. In diesem Eintrag befindet sich der Pfad zu der definierten Mapping-Strategie, in diesem Fall lautet dieser „de.wiemod.virtuos.mapper.AddSubMixMappingStrategy“. Darin wird das in Kapitel 7.1 entwickelte Vorgehen, bei dem entsprechend den Ein- und Ausgängen des gemischten Additions-/Subtraktionsblocks ein Submodell in Virtuos generiert wird, in einer Java-Klasse implementiert. Das Submodell enthält einen Additions- und einen Subraktionsblock. Die Verknüpfung dieser beiden Blöcke erfolgt generisch, je nachdem wie viele Ein- und Ausgänge der gemischte Additions-/Subtraktionsblock in Matlab/Simulink enthält und wie die Reihenfolge der Vorzeichen festgelegt ist (Bild 8-18).

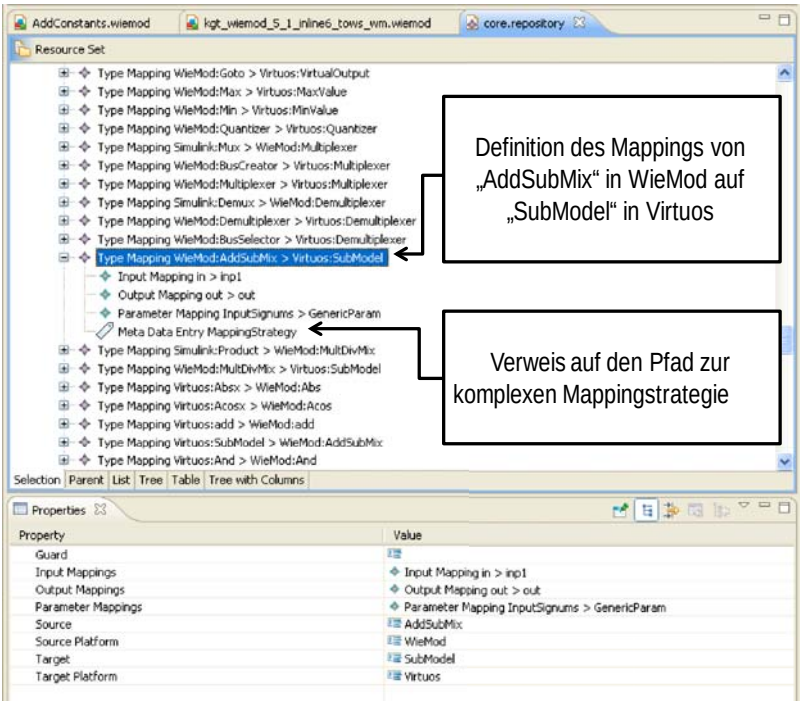


Bild 8-18: Definition des Mappings vom Meta-Metamodell zum Zielwerkzeug

8.2 Validierung

Die Validierung des erarbeiteten Konzepts zur Wiederverwendung von Simulationsmodellen und der entwickelten Methoden erfolgte zunächst anhand des bereits in den vorherigen Kapiteln verwendeten, einfachen Beispielmodells „AddConstants“. Dieses wurde zunächst aus den beiden prototypisch in die WieMod-Workbench integrierten Ausgangswerkzeugen Virtuos (in Form der Virtuos-Modelldatei „AddConstants.ecf“) und Matlab/Simulink (in Form der Matlab/Simulink-Modelldatei „AddConstants.mdl“) in das WieMod Meta-Metamodell transformiert. Die aus der Transformation resultierenden Meta-Metamodelldateien „AddConstants.wiemod“ und „AddConstants.layout“ wurden danach in Modelldateien für die drei prototypisch in die WieMod-Workbench integrierten Zielwerkzeuge Virtuos, Matlab/Simulink und SMP2 transformiert. So konnte die korrekte Funktionsweise der WieMod-Workbench und somit die Wiederverwendbarkeit der jeweiligen Modelldateien aus beiden Ausgangswerkzeugen in jedem der drei Zielwerkzeuge an diesem einfachen Beispielmodell nachgewiesen werden.

In den folgenden Unterkapiteln wird ein Praxisbeispiel aus der Domäne Maschinenbau und der Disziplin Steuerungstechnik ausführlich betrachtet. Dazu wurde das komplexe Modell eines Kugelgewindetriebs ausgewählt. Kugelgewindetriebe werden in Werkzeugmaschinen eingesetzt, um eine rotatorische in eine translatorische Bewegung umzusetzen, z.B. um bei Drehmaschinen den Werkstückträger (Tisch) zu verfahren. Kugelgewindetriebe kommen auch in Luft- und Raumfahrt zum Einsatz. In der Luftfahrt werden Kugelgewindetriebe zum Beispiel für die Landeklappenverstellung von Flugzeugen verwendet, in der Raumfahrt kommen Kugelgewindetriebe beispielweise bei Satelliten zum Einsatz, etwa zum Ausrichten von Solarpanelen. Das ausgewählte Simulationsmodell des Kugelgewindetriebs ist somit ein repräsentatives Beispiel für die domänenübergreifende Einsatzmöglichkeit von Simulationsmodellen und ein mögliches Anwendungsfeld für wiederverwendbare Simulationsmodelle.

Das verwendete Simulationsmodell wurde in Matlab/Simulink erstellt. Matlab/Simulink ist somit in der Validierung das Ausgangswerkzeug. Als Zielwerkzeug wurde Virtuos verwendet. Das ausgewählte Modell enthält mehrere Submodelle, die ihrerseits wiederum Submodelle enthalten. /70/

Anhand dieses Modells soll die WieMod-Workbench hinsichtlich der in Kapitel 5.2 definierten Anforderungen validiert werden.

In Bild 8-19 ist die oberste Ebene des Originalmodells im Ausgangswerkzeug Matlab/Simulink zu sehen. Neben einem Submodell für den eigentlichen Kugelgewindetrieb enthält es zusätzliche Submodelle für den Lageregler, den Drehzahlregler, die Streckennormierung sowie für Motor und Stromregler.

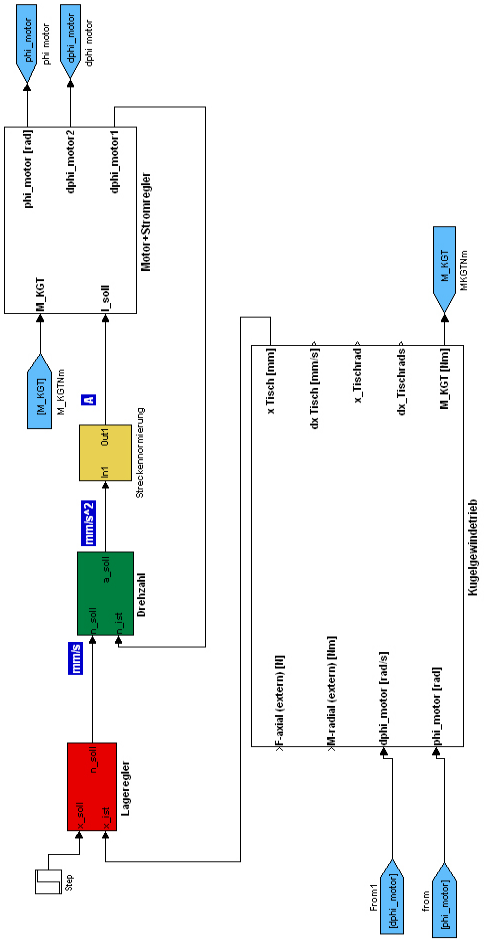


Bild 8-19: Gesamtmodell des Kugelgewindetriebs mit Reglern in Matlab/Simulink /70/

Dieses Modell wurde mit der WieMod-Workbench in das Zielwerkzeug Virtuos transformiert und anschließend dort importiert. Das importierte Modell ist in Bild 8-20 zu sehen.

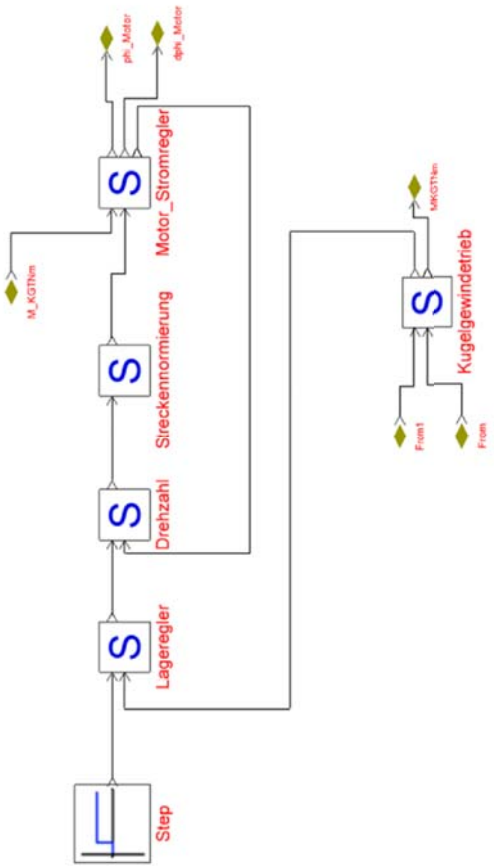


Bild 8-20: Transformiertes Gesamtmodell des Kugelgewindetriebs mit Reglern in Virtuos

Weiterhin werden im Folgenden beispielhaft zwei der mit der WieMod-Workbench von Matlab/Simulink nach Virtuos transformierten Submodelle des Kugelgewindetriebs vorgestellt.

In Bild 8-21 ist das Submodell, das Stromregler und Motor beinhaltet, in Matlab/Simulink dargestellt.

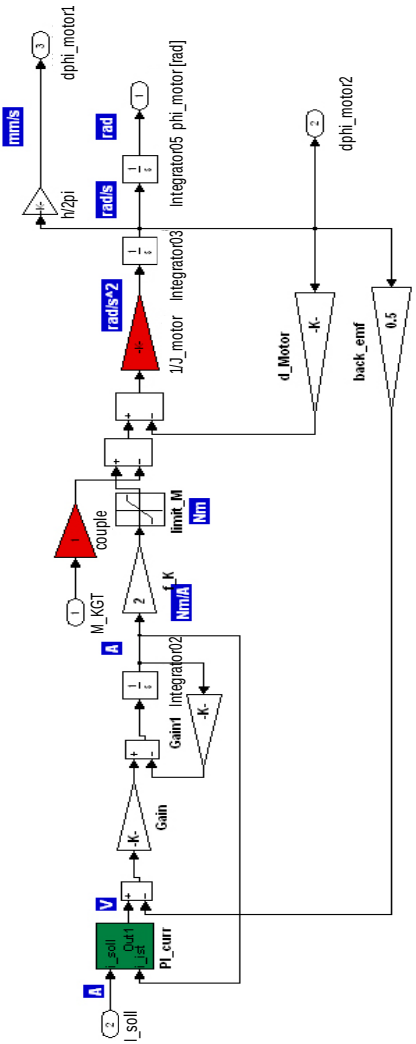


Bild 8-21: Submodell des Stromreglers und des Motors in Matlab/Simulink /70/

In Bild 8-22 ist das transformierte und in Virtuos importierte Submodell von Stromregler und Motor dargestellt.

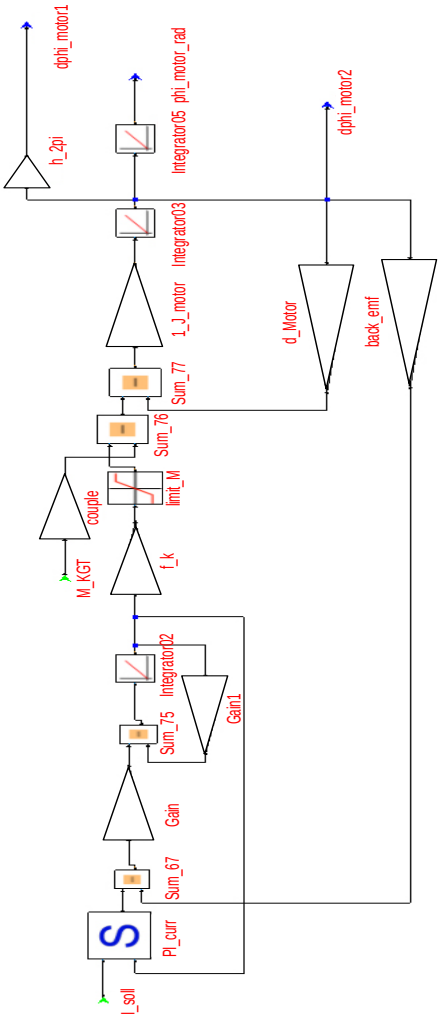


Bild 8-22: Submodell des Stromreglers und des Motors in Virtuos

8.2.1 Validierung hinsichtlich Lesbarkeit

Die Eigenschaft der Lesbarkeit eines Modells bedeutet (wie bereits in Kapitel 5.2.1 definiert), dass das Dateiformat, in dem das Modell abgespeichert ist, zum einen offengelegt und zum anderen interpretierbar ist. Dies ist bei den Modellen aller drei betrachteten Simulationswerkzeuge der Fall. In den Kapiteln 6.2.1 - 6.2.3 wurden die jeweiligen Dateiformate analysiert. Dabei konnte festgestellt werden, dass jedes dieser Dateiformate die Anforderung nach Lesbarkeit erfüllt.

8.2.2 Validierung hinsichtlich Rechenbarkeit

Die Anforderung der Rechenbarkeit muss bei der Modelltransformation von einem Ausgangswerkzeug zu einem Zielwerkzeug erfüllt sein, damit die korrekte Berechnung des Simulationsmodells gewährleistet ist. Dabei dürfen die Ausgabewerte eines bestimmten Signals nach der Transformation nicht von denen des ursprünglichen Modells abweichen. Bei der Transformation werden die Blöcke des Modells aus dem Ausgangswerkzeug sowie alle Signalverbindungen zwischen den Blöcken in das Zielwerkzeug übernommen. Die eingestellten lokalen Parameterwerte der einzelnen Blöcke werden dabei gleichzeitig übertragen.

Verschiedene Simulationswerkzeuge wenden möglicherweise unterschiedliche numerische Lösungsverfahren an. Da dies zu Abweichungen in der Berechnung führen kann, wird das im Ausgangswerkzeug verwendete Lösungsverfahren an das Zielwerkzeug in Form eines globalen Parameters übergeben. Wird das Lösungsverfahren vom Zielwerkzeug nicht unterstützt, erhält der Nutzer einen entsprechenden Hinweis.

Zur Validierung der Erfüllung der Anforderung der Rechenbarkeit anhand des ausgewählten Modells, wird zunächst sowohl im Ausgangs- als auch im Zielwerkzeug die stationäre Reaktion auf einen Einheitssprung, der als Sollwert auf einen Eingang des Lagereglers des Kugelgewindetriebs gegeben wird, untersucht. Die Reaktion des Systems, die Sprungantwort, wird dazu am Ausgang des Kugelgewindetriebs gemessen.

In Bild 8-23 ist die Sprungantwort des Systems im Ausgangswerkzeug Matlab/Simulink auf den vorgegebenen Einheitssprung zu sehen.

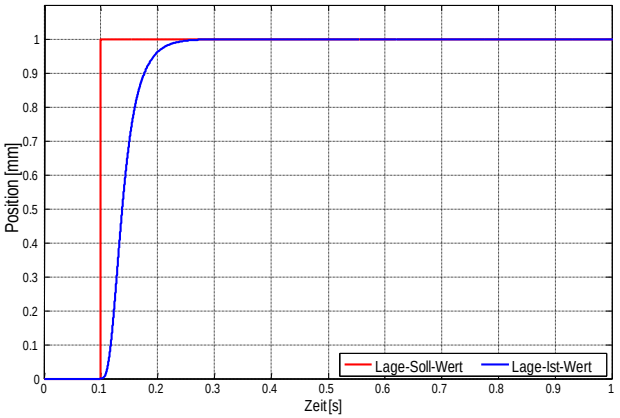


Bild 8-23: Sprungantwort des Systems im Ausgangswerkzeug

In Bild 8-24 ist die Sprungantwort des Systems auf den gleichen Einheitssprung im Zielwerkzeug Virtuos zu sehen.

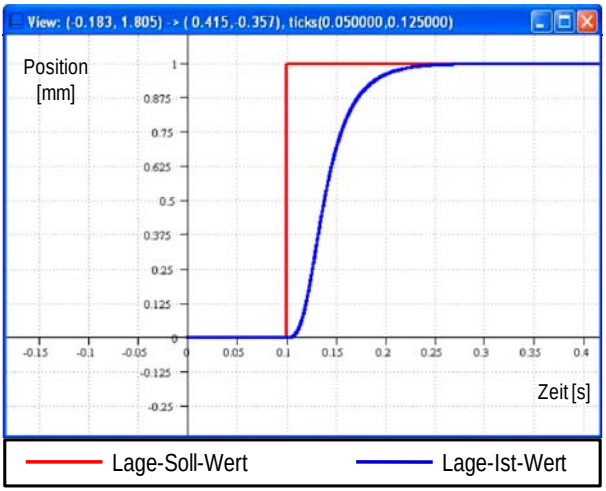


Bild 8-24: Sprungantwort des Systems im Zielwerkzeug

Ein Vergleich zeigt, dass das transformierte Simulationsmodell im Zielwerkzeug bzgl. der Sprungantwort das gleiche Verhalten wie das ursprüngliche Simulationsmodell im Ausgangswerkzeug aufweist.

Um zu validieren, dass das wiederverwendete Simulationsmodell im Zielwerkzeug über einen großen Frequenzbereich das gleiche Verhalten aufweist wie das Originalmodell im Ausgangswerkzeug, wird ein Sinus-Sweep über einen Frequenzbereich von 10-500 Hz in beiden Simulationswerkzeugen als Anregung auf einen Eingang des jeweiligen Systems gegeben. Der Frequenzbereich wurde gewählt, da ab 10 Hz der relevante Bereich beginnt, in dem sich der Kugelgewindetrieb in seinem Verhalten verändert.

In der Signalverarbeitung ist ein Sweep als ein periodisches Signal definiert, das in einer festgelegten Zeit seine Frequenz von einem Startwert f_{Start} zu einem Endwert f_{Stopp} hin ändert. Dabei werden kontinuierlich alle Sinusschwingungen bzw. Frequenzen von f_{Start} bis f_{Stopp} auf den Eingang des zu untersuchenden Systems gegeben. Die Amplitude der Sinusschwingungen bleibt dabei konstant. Am Ausgang hängt die gemessene Amplitude von den frequenzmäßigen Eigenschaften des untersuchten Systems ab. Indirekt stellt die Zeitachse des Sweep-Signals auch eine Frequenzachse von f_{Start} bis f_{Stopp} dar. /71/

Die Reaktion darauf wird jeweils an einem Ausgang des Systems gemessen und in einem Bode-Diagramm dargestellt. In einem Bode-Diagramm werden der Betrag A(ω) und die Phase φ(ω) des Frequenzganges $G(j\omega) = A(\omega)e^{j\varphi(\omega)}$ getrennt über der Frequenz ω aufgetragen. Auf diese Weise werden Amplitudengang bzw. Betragskennlinie und Phasengang bzw. Phasenkennlinie des Übertragungsgliedes gebildet. Beide zusammen ergeben die Frequenzkennlinien-Darstellung. Dabei werden A(ω) und ω logarithmisch und φ(ω) linear aufgetragen. Diese Darstellung wird als Bode-Diagramm bezeichnet. A(ω) wird darin üblicherweise in dB angegeben. /72/

In Bild 8-25 ist zunächst die Anregung in Form der beiden aufeinander folgenden Sinus-Sweeps als Eingangssignal auf den Drehzahlregler zu sehen. Die Reaktion des Systems ist im unteren Diagramm als Ausgangssignal aufgetragen. Eingangs- und Ausgangssignal sind im Zeitbereich dargestellt. Die Einheit der Drehzahl wird in mm/s angegeben, da diese im Modell des Kugelgewindetriebs auf die translatorische Einheit am Abtrieb normiert ist.

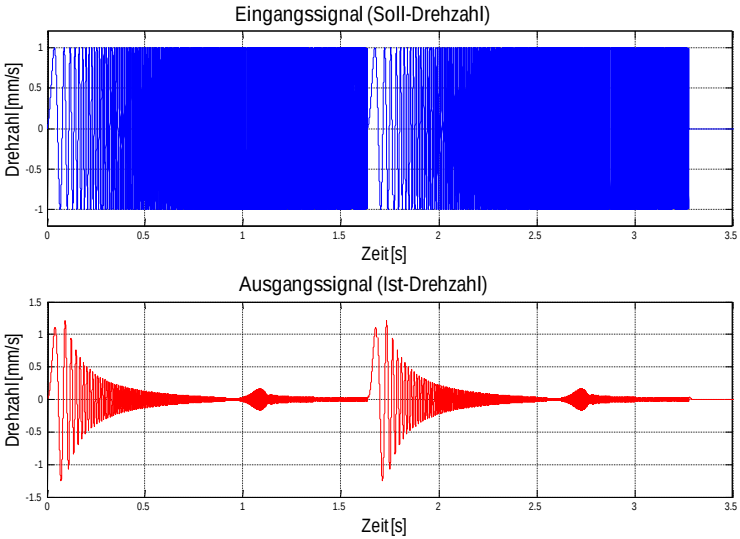


Bild 8-25: Sinus-Sweep

Diese Anregung wird als Sollwert auf den Eingang des Drehzahlreglers des Kugelgewindetriebs gegeben. In Bild 8-26 ist das Bode-Diagramm, das die Reaktion des Modells auf den Sinus-Sweep im Ausgangswerkzeug Matlab/Simulink beschreibt, zu sehen. Das Bode-Diagramm ist die Darstellung von Ein- und Ausgangssignal des Systems im Frequenzbereich.

Virtuos bietet nicht die Möglichkeit, Bode-Diagramme zu generieren. Deshalb wurden die Soll- und Ist-Werte des wiederverwendeten Modells in Virtuos aufgezeichnet und in Matlab ausgewertet. In Bild 8-27 ist das Bode-Diagramm, das die Reaktion des Modells auf den Sinus-Sweep im Zielwerkzeug Virtuos beschreibt, zu sehen.

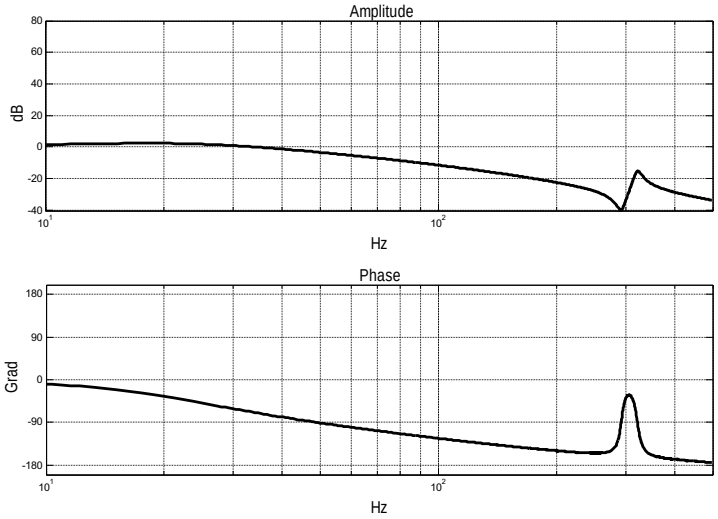


Bild 8-26: Bode-Diagramm des Systemverhaltens im Ausgangswerkzeug Matlab/Simulink

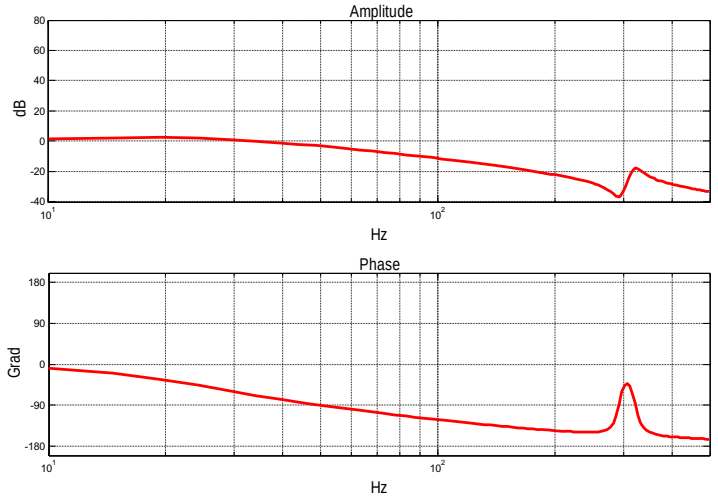


Bild 8-27: Bode-Diagramm des Systemverhaltens im Zielwerkzeug Virtuos

Die markanten Punkte im Amplituden- und Phasengang stimmen für die Modelle in beiden Simulationswerkzeugen überein, wie in Bild 8-28 zu sehen ist. Beide Modelle verhalten sich nahezu gleich. Somit ist die Anforderung nach Rechenbarkeit eindeutig erfüllt.

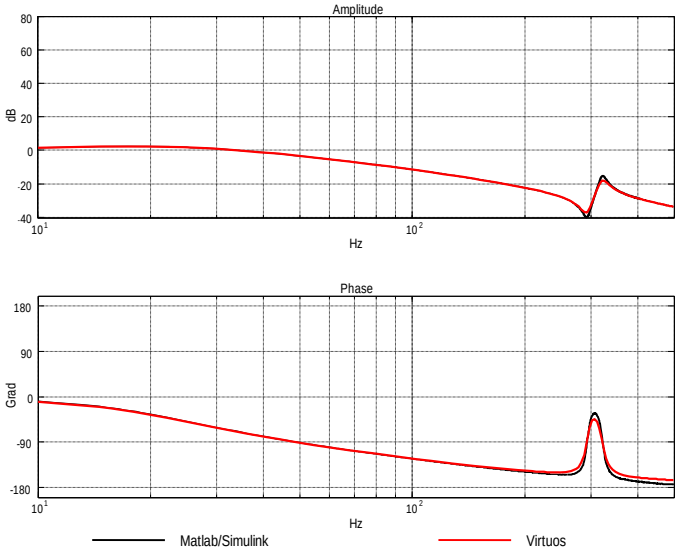


Bild 8-28: Vergleich des Systemverhaltens in Ausgangs- und Zielwerkzeug

8.2.3 Validierung hinsichtlich Wiedererkennbarkeit

Direkt ersichtlich aus den Bildern des Modells im Ausgangswerkzeug sowie den Bildern des transformierten Modells im Zielwerkzeug ist, dass die Topologie, also die Anordnung der Blöcke und deren Verbindungen zueinander identisch sind. Die Blöcke des ursprünglichen Gesamtmodells aus Matlab/Simulink sind in Virtuos in Submodellen maskiert. Die Geometrie der Blöcke wird von der Workbench erfasst und im jeweiligen Zielwerkzeug übernommen. Die Hierarchie des Modells, d.h. dessen Unterteilung in Submodelle ist in Ausgangswerkzeug und Zielwerkzeug identisch, wie in den Bildern der Transformationen des Gesamtmodells (Bild 8-19 und Bild 8-20) und des exemplarisch abgebildeten Submodells (Bild 8-21 und Bild 8-22) eindeutig zu sehen ist. Die Namen der Blöcke werden bei der Transformation ebenfalls übernommen.

Für den Nutzer ist damit auf den ersten Blick ersichtlich, dass es sich um die gleichen Modelle handelt. Dies erhöht die Intuitivität bei der Nutzung der WieMod-Workbench und führt somit zu einer erhöhten Akzeptanz seitens der Anwender. Die Erfüllung der Anforderung nach Wiedererkennbarkeit ist somit ebenfalls belegt.

8.2.4 Validierung hinsichtlich Konfigurierbarkeit

Nach der Transformation besteht im Zielwerkzeug genau wie im Ausgangswerkzeug die Möglichkeit, das Modell zu konfigurieren. Das bedeutet, dass das wiederverwendete Modell im Zielwerkzeug genau so behandelt werden kann, als ob es darin ursprünglich erstellt worden wäre.

Zum einen können globale Parameter, d.h. Parameter, die sich auf das gesamte Modell beziehen, wie Modellname, Rechenschrittweite und Integrationsverfahren etc., verändert werden. Zusätzlich kann der Workspace, das heißt die Arbeitsoberfläche, bezüglich graphischer Repräsentation frei konfiguriert werden. (Bild 8-29)

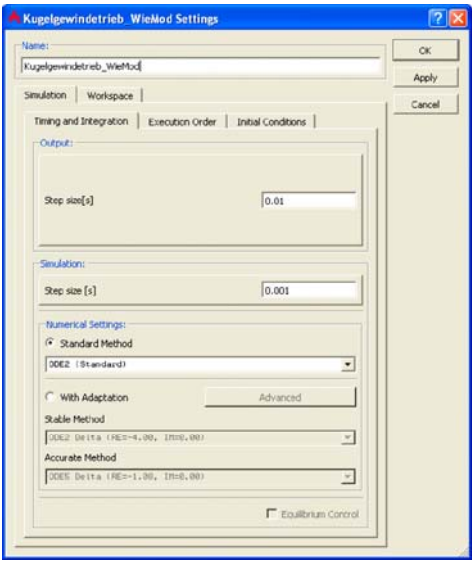


Bild 8-29: Konfigurierbarkeit der globalen Modellparameter

Neben den globalen Parametern können nach der Modelltransformation auch die lokalen Parameter, d.h. die Parameter für einzelne Blöcke, konfiguriert werden. Bild 8-30 zeigt beispielhaft die Parametermaske für einen im Modell enthaltenen Saturation-Block in Virtuos.

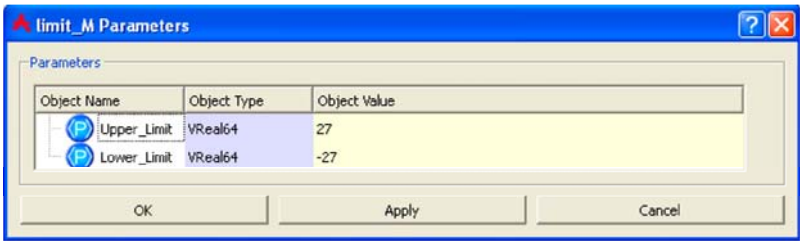


Bild 8-30: Parametermaske zur Konfigurierung lokaler Parameter

Der Saturation-Block begrenzt ein Signal durch eine obere und eine untere Grenze. Der Bereich wird durch die zwei Parameter „UpperLimit“ und „LowerLimit“ vorgegeben. Diese beiden Parameter können in der dargestellten Parametermaske beliebig verändert werden. Für den Nutzer bleibt das wiederverwendete Modell konfigurierbar, obwohl es ursprünglich in einem anderen Simulationswerkzeug erstellt wurde.

Somit ist auch gezeigt, dass die Anforderung nach der Konfigurierbarkeit eines wiederverwendeten Simulationsmodells mit der WieMod-Workbench erfüllt wird.

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Konzept zur domänen- und disziplinübergreifenden Wiederverwendung vorhandener Simulationsmodelle in verschiedenen Simulationswerkzeugen erarbeitet. Die zur Anwendung des Konzepts entwickelten Methoden ermöglichen eine bidirektionale Modelltransformation, durch die der Aufwand zur Wiederverwendung von bestehenden Simulationsmodellen in anderen Simulationswerkzeugen deutlich reduziert wird. So kann sich zukünftig der Aufwand zur Erstellung von Simulationsmodellen früher amortisieren. Damit ist die Simulationstechnik schon in frühen Phasen des Engineerings effizienter nutzbar, so dass letztlich der gesamte Produktentwicklungsprozess davon profitiert.

Zur Konzeption der Wiederverwendung wurden zunächst vorhandene Konzepte aus Forschung und industrieller Praxis im Bereich der Wiederverwendung von Modellen untersucht. Zwei mögliche Vorgehensweisen zur Modelltransformation wurden anschließend analysiert und bezüglich ihrer Effizienz bewertet. Auf Basis dieser Untersuchung wurde die Vorgehensweise der indirekten Modelltransformation über ein Metamodell ausgewählt. Im Folgenden wurden Anforderungen an wiederverwendbare Modelle ermittelt, vor deren Hintergrund schließlich ein Meta-Metamodell entwickelt wurde. Im Rahmen der Entwicklung wurden drei spezifische Simulationswerkzeuge exemplarisch untersucht. Für jedes dieser Simulationswerkzeuge wurde ein werkzeugspezifisches Metamodell entwickelt, das den Aufbau und die Eigenschaften der werkzeugspezifischen Modelle allgemeingültig beschreibt. Das übergeordnete Meta-Metamodell beschreibt wiederum den Aufbau und die Eigenschaften der werkzeugspezifischen Metamodelle.

Durch die in dieser Arbeit entwickelten Methoden kann eine Wiederverwendung einzelner Modellelemente oder ganzer Modelle in verschiedenen Simulationswerkzeugen erfolgen. Dazu wird das gewünschte Modell bzw. Modellelement von der Modellbeschreibungssprache des Ausgangswerkzeugs in die des Zielwerkzeugs transformiert. Nach der Transformation kann das Modell oder das Modellelement des Ausgangswerkzeugs wie gewohnt in dem Zielwerkzeug verwendet werden. Auf diese Weise kann eine vollwertige Wiederverwendung von Simulationsmodellen stattfinden.

Da die Transformation nicht direkt von Ausgangs- zu Zielwerkzeug, sondern über das zwischengeschaltete Meta-Metamodell stattfindet, hat diese Vorgehensweise den entscheidenden Vorteil, dass das erarbeitete System um beliebig viele Simulationswerkzeuge flexibel und mit überschaubarem Aufwand erweitert werden kann. Statt der Implementierung einer neuen Schnittstelle zu jedem bisher integrierten Simulationswerkzeug und zurück müssen bei der Verwendung des Meta-Metamodells nur eine Schnittstelle vom neuen Werkzeug zum Meta-Metamodell und eine zurück vom Meta-Metamodell zum neuen Simulationswerkzeug realisiert werden. Die Anzahl der notwendigen neuen Schnittstellen beim Hinzufügen eines neuen Simulationswerkzeugs beläuft sich somit bei n verwendeten Simulationswerkzeugen lediglich auf $m=2$ statt auf $m=2 \cdot n$, wie es bei der direkten Transformation ohne Meta-Metamodell der Fall wäre.

Das verwendete Meta-Metamodell wurde im Rahmen des vom Bundesministerium für Bildung und Forschung geförderten Projekts „WieMod“ konzipiert und durch die itemis AG implementiert. Das entwickelte Konzept wurde in Form einer Eclipse-basierten Workbench, der WieMod-Workbench, prototypisch umgesetzt. Die WieMod-Workbench ermöglicht den Import von Simulationsmodellen aus den Ausgangswerkzeugen Matlab/Simulink und Virtuos. Diese Modelle werden in das Meta-Metamodell transformiert und können von dort wiederum in Simulationsmodelle für die Zielwerkzeuge Matlab/Simulink, Virtuos und SMP2 transformiert werden.

Zur Umsetzung der Transformationen wurden Mapping-Strategien entwickelt, in denen die Modellelemente eines Simulationsmodells für ein gewähltes Ausgangswerkzeug auf die entsprechenden Modellelemente eines Simulationsmodells für ein gewünschtes Zielwerkzeug abgebildet werden. In der Transformation der Modelle werden die zuvor definierten Mapping-Strategien angewandt, indem die Modellelemente des Ausgangswerkzeugs durch die entsprechenden Modellelemente des Zielwerkzeugs ersetzt werden.

Die entwickelten Methoden sowie deren prototypische Realisierung wurden zunächst anhand eines einfachen Beispielmodells getestet und abschließend am komplexen Modell eines Kugelgewindetriebs aus der Domäne Maschinenbau und der Disziplin Steuerungstechnik validiert. Dabei wurde insbesondere die Erfüllung der eingangs ermittelten Anforderungen an die Wiederverwendbarkeit überprüft.

9.2 Ausblick

Das entwickelte Konzept mit den zugehörigen Methoden trägt maßgeblich zum effizienteren Einsatz der Simulationstechnik im Produktentstehungsprozess bei. Eine konsequente Wiederverwendung und Erweiterung bestehender Simulationsmodelle aus fremden Ausgangswerkzeugen birgt ein hohes Einsparpotenzial im Gegensatz zur Neuerstellung für einen neuen Einsatzbereich im Zielwerkzeug. Die weitere Verbreitung des entwickelten Prototyps sowie dessen Erweiterung durch neue Schnittstellen zu weiteren Simulationswerkzeugen würde ein System mit hohem Nutzen für den Anwender ergeben. Durch die Tatsache, dass der Anwender ein Simulationsmodell aus einem beliebigen Ausgangswerkzeug in seinem gewohnten Zielwerkzeug verwenden kann, ergibt sich eine hohe Akzeptanz sowie eine effiziente Arbeitsweise mit dem wiederverwendeten Modell.

In der prototypischen Umsetzung wurde ein 1:1-Mapping für die Standardblöcke aus den integrierten Simulationswerkzeugen vorgenommen. Verschiedene komplexe Mappings wurden beispielhaft realisiert. Die dazu notwendigen komplexen Mapping-Strategien mussten jedoch individuell für jedes komplexe Modellelement manuell implementiert werden. Dazu ist tiefgehendes Know-How über die Architektur der WieMod-Workbench sowie über die Frameworks, auf denen die WieMod-Workbench basiert, notwendig. Auf Grund dessen können noch nicht alle beliebigen Simulationsmodelle transformiert und direkt wiederverwendet werden. Weiteres Potenzial zur Steigerung der Effizienz des Konzeptes liegt daher in der Integration von mehr komplexen Modellelementen sowie die Automatisierung von komplexen Mapping-Strategien. Um das Konzept dennoch schon erfolgreich einsetzen zu können, müssen aktuell bestimmte Modellierungsrichtlinien eingehalten werden, um eine erfolgreiche Modelltransformation garantieren zu können. Vorhandene Modelle müssen entsprechend angepasst werden, falls sie diesen Modellierungsrichtlinien nicht entsprechen. Dies mindert aktuell noch die Effizienz, da die Anpassung ebenfalls einen zusätzlichen Aufwand bedeutet.

Durch eine produktreife Umsetzung der entwickelten Methoden könnte eine optimale Wiederverwendung vorhandener Simulationsmodelle aus alten und fremden Projekten erzielt werden. Dies setzt voraus, dass die Schnittstellen zu den jeweiligen Simulationswerkzeugen offengelegt und die Modelldateien interpretierbar sind. Die Simulation insgesamt könnte durch die Wiederverwendung von Simulationsmodellen wirtschaftlicher und anwenderfreundlicher gestaltet werden.

10 Literaturverzeichnis

/1/ Eversheim, W.; Schuh, G.: Integrierte Produkt- und Prozessgestaltung. Springer Verlag, Heidelberg, 2005

/2/ Gausemeier, J.; Hahn, A.; Kespohl, H.; Seifert, L.: Vernetzte Produktentwicklung. Der erfolgreiche Weg zum Global Engineering Networking. Hanser Verlag, München, 2006

/3/ Schäppi, B.; Andreasen, M.; Kirchgeorg, M.; Radermacher, J F.: Handbuch Produktentwicklung. Hanser Verlag, München, 2005

/4/ Krause, F-L.; Franke, H-J.; Gausemeier, J.: Innovationspotenziale in der Produktentwicklung. Hanser Verlag, München, 2007

/5/ Bullinger, H-J.; Warnecke, WE H-J.: Neue Organisationsformen im Unternehmen: Ein Handbuch für das moderne Management. Springer Verlag, Heidelberg, 2003

/6/ Gebhardt, C.: Konstruktionsbegleitende Berechnung mit ANSYS DesignSpace: FEM-Simulation für Konstrukteure. Hanser Verlag, München, 2009

/7/ Zülch, G.; Stock, P.: Integrationsaspekte der Simulation: Technik, Organisation und Personal: Karlsruhe, 7. und 8. Oktober 2010; 14. Fachtagung der Arbeitsgemeinschaft Simulation; ASIM-Fachtagung "Simulation in Produktion und Logistik". KIT Scientific Publishing, Karlsruhe, 2010

/8/ Allweyer, T.: Geschäftsprozessmanagement. W3I GmbH, Herdecke, 2005

/9/ Müller, V.; Verl, A.: Wieder verwendbare Modelle zur disziplin- und domänenübergreifenden Simulation. Hybride Technologien in der Produktion. In: Krüger, J.: Hybride Technologien in der Produktion; Fortschritt-Berichte VDI, Reihe 2, Nr. 675. Düsseldorf, VDI-Verlag, 2010, S. 9-21

/10/ Verl, A.; Haubelt, A.; Müller, V.: Baukastenbasiertes simulationsgestütztes Engineering. Ein Lösungsansatz für den durchgängigen Produktentwicklungsprozess. In: A&D-Kompendium 2009/2010. München, publish-industry Verlag GmbH, 2009, S. 72-74

/11/ Engelhardt-Nowitzki, C.; Nowitzki, O.; Krenn, B.: Praktische Anwendung der Simulation im Materialflussmanagement: Erfolgsfaktoren und Implementierungsszenarien. Gabler Verlag, Wiesbaden, 2008

/12/ Müller, V.; Verl, A.: Analysis of different approaches to the reuse of simulation models for the virtual knowledge-based product development. In: Proc. of Conference on Grand Challenges in Modeling & Simulation (GCMS10). Ottawa, 2010

/13/ ISO TC184/SC4/WG7N 393: Industrial Automation Systems and Integration, Part 23. Berlin, Beuth Verlag, 1995

/14/ VDI 3633: Simulation von Logistik-, Materialfluß- und Prodiuktionssystemen. Verein Deutscher Ingenieure, Düsseldorf, 2000

/15/ Strahringer, S.: Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysemethoden.

Shaker Verlag, Aachen, 1996

/16/ Brugger, R.: IT-Projekte strukturiert realisieren. Situationen analysieren, Lösungen konzipieren - Vorgehen systematisieren, Sachverhalte visualisieren - UML und EPKs nutzen. Springer Fachmedien, Wiesbaden, 2005

/17/ Cellier, E. F.: Continuous System Modeling. Springer Verlag, New York, 1991

/18/ Bungartz, H-J.: Modellbildung und Simulation. Springer Verlag, Berlin, 2009

/19/ IEEE Std. 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers, New York, 1990

/20/ Rezagholi, M.: Management der Wiederverwendung in der Softwareentwicklung. In: WIRTSCHAFTSINFORMATIK, Heft 3. Wiesbaden, Gabler Verlag, 1995, S. 221-230

/21/ Eichinger, R.; Leitner, T.; Reisinger, E.: Management der Wiederverwendung von Software, Seminar aus Software Engineering. Linz, Institut für Wirtschaftsinformatik der Johannes Kepler Universität Linz, Arbeitsgruppe Software Engineering, 1999, S. 11-12

/22/ Rothe, A.: Systematische Wiederverwendung von Softwarekomponenten bei Finanzdienstleistern (Diss.). Fakultät Wirtschafts- und Sozialwissenschaften der Universität Stuttgart, Stuttgart, 2004

/23/ Schäling, B.: Der moderne Softwareentwicklungsprozess mit UML
http://www.highscore.de/uml/ (03.06.2012)

/24/ Weilkiens, T.: Systems Engineering mit SysML/UML.
dpunkt Verlag, Heidelberg, 2008

/25/ Korff, A.: Modellierung von eingebetteten Systemen mit UML und SysML.
Spektrum Akademischer Verlag, Heidelberg, 2008

/26/ Hundt, L.: AutomationML
https://www.automationml.org (03.06.2012)

/27/ Fritzson, PA.: Principles of Object-oriented Modeling and Simulation with Modelica 2.1.
John Wiley and Sons, Hoboken, 2004

/28/ N.N.: OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2
http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF (03.06.2012)

/29/ N.N.: OMG's Meta Object Facility (MOF)
http://www.omg.org/mof (03.06.2012)

/30/ N.N.: Model Bus
http://www.modelbus.org/modelbus/current/ModelBus_UserGuide_097_v1.9.7.pdf (03.06.2012)

/31/ Litto, M. et al.: Baukastenbasiertes Engineering mit Föderal - Ein Leitfaden für Maschinen- und Anlagenbauer.
VDMA Verlag, Frankfurt a. M., 2004

/32/ Scheifele, D.; Anwendungen der Hardware-in-the-Loop Simulation in der Produktionstechnik.

Müller, V.: In: SPS/IPC/Drives, 25.-27. Nov. 2008, Nürnberg. Berlin, VDE Verlag GmbH, 2008, S. 377-386

/33/ Verl, A.; Haubelt, A.: Der Weg zur automatischen Generierung von Simulationsmodellen aus mechatronischen Baukästen. In: Brecher, C.: Fortschritt-Berichte VDI, Reihe 2, Fertigungstechnik. Aachen, VDI Verlag, 2008

/34/ Buck, R.: AQUIMO - Ein Leitfaden für Maschinen- und Anlagenbauer. Frankfurt a. M., VDMA-Verlag, 2010

/35/ Klemm, P.; Rüdele, H.; Weimer, T.: Durchgängiges Engineering von mechatronischen Systemen. In: Fertigungstechnisches Kolloquium Stuttgart (FTK 2006)., Gesellschaft für Fertigungstechnik, 2006, S. 101–131

/36/ Reuter, A.; Müller, V.; Verl, A.: Disziplinübergreifendes Engineering. Integration von Simulationsdaten in mechatronische Komponentenmodelle. In: wt Werkstattstechnik online 100 (2010) Nr. 5., S. 399-406

/37/ ISO 10303-21: Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und Austausch. Beuth Verlag, Berlin, 2002

/38/ Anderl, R.; Trippner, D.: STEP STandard for the Exchange of Product Model Data, Eine Einführung in die Entwicklung, Implementierung und industrielle Nutzung der Normenreihe ISO 10303 (STEP). Teubner, 2000

/39/ N.N.: PROSTEP: IntRif http://www.prostep.org/de/projektgruppen/internationalization-of-the-requirements-interchange-format-intrif.html (03.06.2012)

/40/ N.N.: PROSTEP: PSI 6 - Requirements Interchange Format (RIF) http://www.prostep.org/fileadmin/freie_downloads/Empfehlungen-standards/ProSTEP_iViP/PSI_RIF_6_1.2.pdf (03.06.2012)

/41/ Drath, R.; Fedai, M.: CAEX - ein neutrales Datenaustauschformat für Anlagendaten - Teil 1. In: atp - Automatisierungstechnische Praxis. München, R. Oldenbourg Verlag, 2004

/42/ Geisberger, E.; Schmidt, R.: ProMiS – Projektmanagement für interdisziplinäre Systementwicklungen. VDMA-Verlag, Frankfurt a. M., 2004

/43/ N.N., : Co-Simulation - Overview http://www.flowmaster.com/flowmaster_cosimulation.html (03.06.2012)

/44/ Pelz, G.: Modellierung und Simulation mechatronischer Systeme. Hüthig Verlag, Heidelberg, 2001

/45/ Miegel, T.: Funktionale Komponenten eines Federation http://ifgi.uni-muenster.de/~bernard/public/InteropArchitekturen2003/HLA_Ausarbeitung.pdf (03.06.2012)

/46/ ISG: Dokumentation zu Virtuos der Industriellen Steuerungstechnik GmbH. Firmenschrift, Stuttgart, 2011

/47/ Xu, L.: Wiederverwendbare Modelle zur Maschinensimulation für den Steuerungstest. Herbert Utz Verlag, München, 2003

/48/ Henning, K.; Informatik im Maschinenbau.

Kutscha, S.: Verlagsgruppe Mainz in Aachen, Aachen, 2003

/49/ Kersken, S.: IT-Handbuch für Fachinformatiker. Der Ausbildungsbegleiter. Galileo Computing, Bonn, 2011

/50/ Röck, S.: Berichte aus dem ISW, Echtzeitsimulation von Produktionsanlagen mit realen Steuerungssystemen (Diss.). Jost-Jetter Verlag, Heimsheim, 2007

/51/ Furlan, P.: Das gelbe Rechenbuch 3 für Ingenieure, Naturwissenschaftler und Mathematiker. Verlag Martina Furlan, Dortmund, 1995

/52/ DIN EN ISO 9241-110: Ergonomie der Mensch-System-Interaktion - Teil 110: Grundsätze der Dialoggestaltung. Beuth Verlag, Berlin, 2006

/53/ Tremp, H.; Ruggiero, M.: Application Engineering. Grundlagen für die objektorientierte Softwareentwicklung mit zahlreichen Beispielen, Aufgaben und Lösungen. Zürich, Compendio Bildungsmedien, 2011, S. 135

/54/ Haas, F.: Effizienztreiber innovativer Prozesse. Anwendung der Data Envelopment Analysis am Beispiel der elektronischen C-Teile-Beschaffung (Diss.). Wiesbaden, Dt. Univ.-Verl. (Gabler Edition Wissenschaft), 2004, S. 244

/55/ N.N.: Duden online http://www.duden.de/suchen/dudenonline/Konfiguration (03.06.2012)

/56/ Schwertfeger, B.; Feltes, I.; Berres, A.; Berlin, Modellgetriebene Entwicklung von wieder verwendbaren Modellen. In: Proc. on Software Engineering 2010, Effiziente

M.; Gerndt, A.: Softwarelösungen für komplexe Geschäftsanforderungen. Paderborn, 2010

/57/ Lutz, H.; Wendt, W.: Taschenbuch der Regelungstechnik: Mit MATLAB und Simulink. Wissenschaftlicher Verlag Harri Deutsch, Frankfurt am Main, 2007

/58/ ISG: Spezifikation ECF. Firmenschrift, Stuttgart, 2008

/59/ European Space Agency : SMP 2.0 Metamodel http://taste.tuxfamily.org/wiki/images/1/19/SMP_2.0_Metamodel_-_1.2.pdf (03.06.2012)

/60/ Holzner, S.: Eclipse. O'Reilly Verlag, Köln, 2004

/61/ Voß, M.: Komponentenbasierte Entwicklung von NC-Steuerungssystemen (Diss.). Apprimus Verlag, Aachen, 2011

/62/ Shavor, S.: Eclipse - Open Source Library. Pearson Deutschland GmbH, München, 2004

/63/ Knebel, B.: Data Profiling Mit Eclipse: Von Den Grundlagen Zum Prototypen. Diplomica Verlag, Hamburg, 2009

/64/ Pree, W.: Komponentenbasierte Softwareentwicklung mit Frameworks. dpunkt Verlag, Heidelberg, 1997

/65/ IBM Corporation: EObject (EMF Javadoc) http://download.eclipse.org/modeling/emf/emf/javadoc/2.4.3/or

g/eclipse/emf/ecore/EObject.html (03.06.2012)

/66/ Eclipse Foundation: Eclipse Modeling
http://www.eclipse.org/modeling/emf/ (03.06.2012)

/67/ Eclipse Foundation: Eclipse Help - Xtend/ Xpand/ Check
http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.xpand.doc/help/ch01s05.html (03.06.2012)
2011

/68/ Tanasa, S.; Andrei, S.; Olaru, C.: Java from 0 to expert level.
Polirom Publishing Press, Rumänien, 2007

/69/ Vogel, L.: Eclipse Modeling Framework (EMF) - Tutorial
http://www.vogella.de/articles/EclipseEMF/article.html
Juni 2011

/70/ Croon, N.: Neues KGT Vorschubsystem. Aktueller Stand der Forschung.
17. Lagereglseminar '08, Stuttgart, 2008

/71/ Karrenberg, U.: Signale-Prozesse-Systeme: Eine Multimediale und Interaktive Einführung in die Signalverarbeitung.
Springer Verlag, Berlin, 2009

/72/ Unbehauen, H.: Regelungstechnik I.
Vieweg Verlag, Wiesbaden, 1982

Lebenslauf

Persönliches	Name	Verena Voß
	Geburtsname	Müller
	Geburtstag	27.09.1981
	Geburtsort	Düren-Birkesdorf
Schulbildung	1987 - 1988	Keeneyville School, Roselle, IL, USA
	1988 - 1990	Waterbury School, Roselle, IL, USA
	1990 - 1991	Gemeinschaftsgrundschule Vossenack
	1991-2000	Privates Franziskus Gymnasium Vossenack
Studium	2000 - 2008	RWTH Aachen (Maschinenbau mit Vertiefungsrichtung Produktionstechnik)
Berufstätigkeit	2008 - 2012	Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen, Universität Stuttgart
	Seit 2012	Asys Automatic Systems GmbH & Co. KG, Schorndorf

ISW/IPA Forschung und Praxis

Berichte aus dem Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart und dem Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA

Herausgegeben bis Band 57 von Prof. Dr.-Ing. G. Stute †
ab Band 58 von Prof. Dr.-Ing. Dr. h.c. mult. Dr.-Ing. E.h. G. Pritschow
ab Band 161 von Prof. Dr.-Ing. Dr. h.c. mult. A. Verl

Erschienen bei Springer-Verlag:

1	Schmid, D.:	Numerische Bahnsteuerung, 1973.
2	Schwegler, H.:	Fräsbearbeitung gekrümmter Flächen, 1972.
3	Eisinger, J.:	Numerisch gesteuerte Mehrachsenfräsmaschinen, 1972.
4	Nann, R.:	Rechnersteuerung von Fertigungseinrichtungen, 1972.
5	Augsten, G.:	Zweiachsige Nachformeinrichtungen, 1972.
6	Karl, B.:	Die Automatisierung der Fertigungsvorbereitung durch NC-Programmierung, 1972.
7	Eitel, H.:	NC-Programmiersystem, 1973.
8	Knorr, E.:	Numerische Bahnsteuerung zur Erzeugung von Raumkurven auf rotationssymetrischen Körpern, 1973.
9	Bumiller, S.:	Viskohydraulischer Vorschubantrieb, 1974.
10	Maier, K.:	Grenzregelung an Werkzeugmaschinen, 1974.
11	Waelkens, J.:	NC-Programmierung, 1974.
12	Bauer, E.:	Rechnerdirektsteuerung von Fertigungseinrichtungen, 1975.
13	König, H.:	Entwurf und Strukturtheorie von Steuerungen für Fertigungseinrichtungen, 1976.
14	Damsohn, H.:	Fünfachsiges NC-Fräsen, 1976.
15	Jetter, H.:	Programmierbare Steuerungen, 1976.
16	Henning, H.:	Fünfachsiges NC-Fräsen gekrümmter Flächen, 1976.
17	Boelke, K.:	Analyse und Beurteilung von Lagesteuerungen für numerisch gesteuerte Werkzeugmaschinen, 1977.

18 Götz, F.-R.: Regelsystem mit Modellrückkopplung für variable Streckenverstärkung, 1977.

19 Tränkle, H.: Auswirkungen der Fehler in den Positionen der Maschinenachsen beim fünfachsigen Fräsen, 1977.

20 Stof, P.: Untersuchungen über die Reduzierung dynamischer Bahnabweichungen bei numerisch gesteuerten Werkzeugmaschinen, 1978.

21 Wilhelm, R.: Planung und Auslegung des Materialflusses flexibler Fertigungssysteme, 1978.

22 Kappen, N.: Entwicklung und Einsatz einer direkten digitalen Grenzregelung für eine Fräsmaschine mit CNC, 1979.

23 Klug, H. G.: Integration automatisierter technischer Betriebsbereiche, 1978.

24 Binder, D.: Interpolation in numerischen Bahnsteuerungen, 1979.

25 Klingler, O.: Steuerung spanender Werkzeugmaschinen mit Hilfe von Grenzregeleinrichtungen, 1979.

26 Schenke, L.: Auslegung einer technologisch-geometrischen Grenzregelung für die Fräsbearbeitung, 1979.

27 Wörn, H.: Numerische Steuersysteme – Aufbau und Schnittstellen eines Mehrprozessorsteuersystems, 1979.

28 Osofisan, P. B.: Verbesserung des Datenflusses beim fünfachsigen NC-Fräsen, 1979.

29 Berner, J.: Verknüpfung fertigungstechnischer NC-Programmiersysteme, 1979.

30 Böbel, K.-H.: Rechnerunterstützte Auslegung von Vorschubantrieben, 1979.

31 Dreher, W.: NC-gerechte Beschreibung von Werkstücken in fertigungstechnisch orientierten Programmiersystemen, 1980.

32 Schurr, R.: Rechnerunterstützte Projektsteuerung hydrostatischer Anlagen, 1981.

33 Sielaff, W.: Fünfachsiges NC-Umfangfräsen verwundener Regelflächen. Beitrag zur Technologie und Teileprogrammierung, 1981.

34 Hesselbach, J.: Digitale Lageregelung an numerisch gesteuerten Fertigungseinrichtungen, 1981.

35 Fischer, P.: Rechnerunterstützte Erstellung von Schaltplänen am Beispiel der automatisierten Hydraulikplanzeichnung, 1981.

36 Ackermann, U.: Rechnerunterstützte Auswahl elektrischer Antriebe für spanende Werkzeugmaschinen, 1981.

37 Döttling, W.: Flexible Fertigungssysteme – Steuerung und Überwachung des Fertigungsablaufs, 1981.

38 Firnau, J.: Flexible Fertigungssysteme – Entwicklung und Erprobung eines zentralen Steuersystems, 1982.

39 Herrscher, A.: Flexible Fertigungssysteme – Entwurf und Realisierung prozeßnaher Steuerungsfunktionen, 1982.

40 Spieth, U.: Numerische Steuersysteme – Hardwareaufbau und Ablaufsteuerung eines Mehrprozessorsteuersystems, 1982.

41 Schimmele, A.: Rechnerunterstützter Entwurf von Funktionssteuerungen für Fertigungseinrichtungen, 1982.

42 Sanzenbacher, M.: NC-gerechte Beschreibung von Werkstücken mit gekrümmten Flächen, 1982.

43 Walter, W.: Interaktive NC-Programmierung von Werkstücken mit gekrümmten Flächen, 1982.

44 Huan, J.: Bahnregelung zur Bahnerzeugung an numerisch gesteuerten Werkzeugmaschinen, 1982.

45 Erne, H.: Taktile Sensorführung für Handhabungseinrichtungen – Systematik und Auslegung der Steuerungen, 1982.

46 Plasch, D.: Numerische Steuersysteme – Standardisierte Softwareschnittstellen in Mehrprozessor-Steuersystemen, 1983.

47 Wang, Z. L.: NC-Programmierung – Maschinennaher Einsatz von fertigungstechnisch orientierten Programmiersystemen, 1983.

48 Schwager, J.: Diagnose steuerungsexterner Fehler an Fertigungseinrichtungen, 1983.

49 Klemm, P.: Strukturierung von flexiblen Bediensystemen für numerische Steuerungen, 1984.

50 Runge, W.: Simulation des dynamischen Verhaltens elektrohydraulischer Schaltungen – Einsatz von geräteorientierten, universellen Simulationsbausteinen, 1984.

51 Steinhilber, H.: Planung und Realisierung von Werkzeugversorgungssystemen für die NC-Bearbeitung, 1984.

52 Ohnheiser, R.: Integrierte Erstellung numerischer Steuerdaten für flexible Fertigungssysteme, 1984.

53 Keppeler, M.: Führungsgrößenerzeugung für numerisch bahngesteuerte Industrieroboter, 1984.

54 Kohler, P.: Automatisiertes Messen mit NC-Werkzeugmaschinen, 1985.

55 Rieger, K.-H.: Rechnerunterstützte Projektierung der Hardware und Software von Speicherprogrammierten Steuerungen, 1985.

56 Vogt, G.: Digitale Regelung von Asynchronmotoren für numerisch gesteuerte Fertigungseinrichtungen, 1985.

57 Chmielnicki, S.: Flexible Fertigungssysteme – Simulation der Prozesse als Hilfsmittel zur Planung und zum Test von Steuerprogrammen, 1985.

58 Renn, W.: Struktur und Aufbau prozeßnaher Steuergeräte zur Verkettung in flexiblen Fertigungssystemen, 1986.

59 Harig, K.: Quantisierung im Lageregelkreis numerisch gesteuerter Fertigungseinrichtungen, 1986.

60 Frank, H.: Programmier- und Überwachungsfunktionen für teileartbezogene NC-Werkzeugmaschinen, 1986.

61 Möller, H.: Integrierte Überwachungs- und Diagnose-Systeme für numerische Steuerungen, 1986.

62 Fink, H.: Einsatz speicherprogrammierbarer Steuerungen in der Fertigungstechnik, 1986.

63 Fleckenstein, J.: Zustandsgraphen für SPS – Grafikunterstützte Programmierung und steuerungsunabhängige Darstellung, 1987.

64 Wagner, E.: Steuerungen von Koordinatenmeßgeräten mit schaltenden und messenden Tastsystemen, 1987.

65 Grimm, W.: Diagnosesystem für steuerungsperiphere Fehler an Fertigungseinrichtungen, 1987.

66 Swoboda, W.: Digitale Lageregelung für Maschinen mit schwach gedämpften schwingungsfähigen Bewegungsachsen, 1987.

67 Gruhler, G.: Sensorgeführte Programmierung bahngesteuerter Industrieroboter, 1987.

68 Walker, B.: Konfigurierbarer Funktionsblock Geometriedatenverarbeitung für numerische Steuerungen, 1987.

69 Mayer, J.: Werkzeugorganisation für flexible Fertigungszellen und -systeme, 1988.

70 Lederer, R.: Programmierung von NC-Drehmaschinen mit mehreren Werkzeugschlitten, 1988.

71 Häberle, G.: NC-Musterprogrammierung für rechnerintegrierte Textilfertigung, 1988.

72 Pfeiffer, D.: Kompensation thermisch bedingter Bearbeitungsfehler durch prozeßnahe Qualitätsregelung, 1988.

73 Schmidt, W.: Grafikunterstütztes Simulationssystem für komplexe Bearbeitungsvorgänge in numerischen Steuerungen, 1988.

74 Egner, M.: Hochdynamische Lageregelung mit elektrohydraulischen Antrieben, 1988.

75 Schittenhelm, W.: Konfigurierbares Bedienungssystem für Steuerungen an Fertigungseinrichtungen, 1988.

76 Scheifele, D.: Grafisch dynamische Simulation des Bearbeitungsvorgangs für Doppelschlittendrehmaschinen, 1988.

77 Keuper, G.: Automatisierte Identifikation der Streckenparameter servohydraulischer Vorschubantriebe, 1989.

78 Kayser, K.-H.: Kollisionserkennung in numerischen Steuerungen mit der Distanzfeldmethode, 1989.

79 Viefhaus, R.: Fräsergeometriekorrektur in Numerischen Steuerungen für das fünfachsige Fräsen, 1989.

80 Zirbs, J.: Fertigungsgerechte Aufbereitung von Flächenverbänden bei der NC-Programmierung im Formenbau, 1989.

81 Ruoff, W.: Optische Sensorsysteme zur On-line-Führung von Industrierobotern, 1989.

82 Jantzer, M.: Bahnverhalten und Regelung fahrerloser Transportsysteme ohne Spurbindung, 1990.

83 Schumacher, H.: Einheitliche Programmierung von Automatisierungskomponenten roboterbestückter Bearbeitungs- und Montagezellen, 1991.

84 Schimonyi, J.: NC-Programmierung für das Werkzeugschleifen, 1991.

85 Wurst, K.-H.: Flexible Robotersysteme – Konzeption und Realisierung modularer Roboterkomponenten, 1991.

86 Hagl, R.: Erhöhung der Verfügbarkeit von Vorschubantrieben mit selbstanpassender Lageregelung, 1991.

87 Krebser, G.: Betriebssystem für NC mit einheitlichen Schnittstellen, 1992.

88 Lei, W.-T.: Flächenorientierte Steuerdatenaufbereitung für das fünfachsige Fräsen, 1992.

89 Diehl, G.: Steuerungsperipheres Diagnosesystem für Fertigungseinrichtungen auf Basis überwachungsgerechter Komponenten, 1992.

90 Nepustil, U.: Offene NC-Schnittstellen zur Korrektur von Fertigungsfehlern, 1992.

91 Bauder, M.: Konfigurierbare Robotersteuerung mit allgemeiner Transformation, 1992.

92 Philipp, W.: Regelung mechanisch steifer Direktantriebe für Werkzeugmaschinen, 1992.

93 Härdtner, G. M.: Wissensstrukturierung in Diagnoseexpertensystemen für Fertigungseinrichtungen, 1992.

94 Wiedmann, H.: Objektorientierte Wissensrepräsentation für die modellbasierte Diagnose an Fertigungseinrichtungen, 1993.

95 Rudloff, H.: Hochgenaue Konturerzeugung bei Bewegungsachsen mit einer dominanten mechanischen Resonanzstelle, 1993.

96 Brantner, K.: Adaptierbares Leitsteuerungssystem für flexible Produktionssysteme, 1993.

97 Kugler, W.: Kommunikationsmechanismen für offene Numerische Steuerungssysteme, 1994.

98 Schnurr, B.: Elektrodynamisches Antriebssystem zur Unrundbearbeitung, 1994.

99 Schneider, J.: Fehlerreaktion mit Speicherprogrammierbaren Steuerungen – ein Beitrag zur Fehlertoleranz, 1994.

100 Siewert, U.: Systematische Erstellung adaptierbarer Leitsteuerungssoftware am Beispiel der Durchsetzungsplanung, 1994.

101 Heger, G. F. J.: Maschinenferner Qualitätsregelkreis in flexiblen Fertigungssystemen, 1994.

102 Hofmeister, W.: Objektorientiert strukturiertes Programmiersystem für NC-Mehrschlittenmaschinen, 1994.

103 Horn, A.: Optische Sensorik zur Bahnführung von Industrierobotern mit hohen Bahngeschwindigkeiten, 1994.

104 Rentschler, U.: Fehlertolerantes Präzisionsfügen, 1995.

105 Junghans, G.: Modulares grafikunterstütztes Simulationssystem für Bearbeitungs- und Handhabungsvorgänge, 1995.

106 Heller, J.: Sensorgestützte Bewegungserzeugung leitlinienloser Transprtfahrzeuge, 1995.

107 Wieland, E.: Anwendungsorientierte Programmierung für die robotergestützte Montage, 1995.

108 Ketterer, G.: Automatisierte Inbetriebnahme elektromechanischer, elastisch gekoppelter Bewegungsachsen, 1995.

109 Reibetanz, Th.: Situationsorientierte Bearbeitungsmodellierung zur NC-Programmierung, 1995.

110	Frager, O.:	Durchgängige Programmierung von Fertigungszellen, 1996.
111	Ordenewitz, R.:	Betriebsweite Bereitstellung von Werkzeuginformationen, 1996.
112	Daniel, C.:	Dynamisches Konfigurieren von Steuerungssoftware für offene Systeme, 1996.
113	Angerbauer, R.:	Anwenderorientierte Programmierung fahrerloser Transportsysteme, 1996.
114	Krauß, F.:	Splineverarbeitung in numerischen Steuerungen für das fünfachsige Fräsen, 1996.
115	Schittenhelm, K.-M.:	Einsatz vorgefilterter Führungsgrößen für Bewegungsachsen zur Bahnerzeugung, 1997.
116	Häberle, U.:	Einheitliche Anwenderschnittstelle für Feldbussysteme, 1997.
117	Strassacker, D.:	Testumgebung für die Implementierung und Inbetriebnahme eines adaptierbaren Leitsteuerungssystems, 1997.
118	Renz, B.:	Hochdynamische Strahllagekorrektursysteme zur Erhöhung der Bahngenauigkeit von CO_2-Laserbearbeitungsmaschinen, 1997.
119	Itterheim, C.:	Objektorientiertes Bearbeitungsmodell für Freiformflächen – Erstellung und maschinengebundene Modifikation –, 1997.
120	Müller, J.:	Objektorientierte Softwareentwicklung für offene numerische Steuerungen, 1997.
121	Glöckler, M.:	Verbesserung des Störverhaltens elektrohydraulischer lagegeregelter Zylinderantriebe, 1998.
122	Uhl, J.:	Entwurfssystematik für ein dezentral strukturiertes, objektorientiertes Fertigungsleitsystem, 1998.
123	Hammann, G.:	Modellierung des Abtragsverhaltens elastischer, robotergeführter Schleifwerkzeuge, 1998.
124	Scholich-Tessmann, W.:	Direktantriebe für Industrieroboter, 1998.
125	Wagner, R.:	Robotersysteme zum kraftgeführten Entgraten grobtolerierter Leichtmetallwerkstücke mit Fräswerkzeugen, 1998.
126	Anders, C.:	Adaptierbares Diagnosesystem bei Transferstraßen, 1998.
127	Fahrbach, C.:	Regelung hochdynamischer elektrischer Servo-Direktantriebe in Fertigungseinrichtungen, 1999.
128	Sperling, W.:	Modulare Systemplattformen für offene Steuerungssysteme, 1999.

Erschienen bei Jost-Jetter Verlag:

129 Kehl, G.: Gestaltung von Formgedächtnis-Aktorsystemen für sensorgeführte Inspektionsgeräte, 1999.

130 Brandl, Th.: Anlageninformationssystem - Informationsmodell und Erstellungssystematik, 1999.

131 Gronbach, H.: Simulationswerkzeug für die Gestaltung modularer CO_2-Laserbearbeitungsmaschinen, 1999.

132 Lutz, R.: Softwaretechnik für Maschinennahe Steuerungsfunktionen bei Fertigungseinrichtungen, 1999.

133 Tran, T. L.: Allgemeine Transformation für Maschinen mit Parallelkinematiken, 2000.

134 Bretschneider, J.: Reglerselbsteinstellung für digital geregelte, elektromechanische Antriebssysteme an Werkzeugmaschinen, 2000.

135 Schoenenberg, M.: Zuverlässiger Fertigungsprozess bei Transferstraßen durch präventive Maßnahmen, 2000.

136 Uhl, A.: Flexibles Telerobotersteuerungssystem auf der Basis offener numerischer Steuerungen, 2000.

137 Rui Li: Agentenbasierte NC-Planung für die Komplettbearbeitung auf Dreh-/Fräszentren, 2001.

138 Wildermuth, D.: Bahnvorbereitung in numerischen Steuerungen für Parallelkinematiken, 2001.

139 Handel, D.: Werkergerechte NC-Programmierung zur Komplett-Schleifbearbeitung von Bohrwerkzeugen, 2001.

140 Kosiedowski, U.: Adaptive Vorsteuerverfahren für elektromechanische Bewegungsachsen an Werkzeugmaschinen, 2001.

141 Haug, K.: Laser-Lichtschnittsensorik für die Automatisierung von Metall-Schutzgasschweißprozessen, 2002.

142 Hohenadel, J.: Einheitliches Steuerungssystem für NC und RC, 2002.

143 Wälde, K.: Sicherstellung der Softwarequalität von Anwendungsmodulen und Systemplattformen in offenen Steuerungssystemen, 2002.

144 Litto, M.: Störungsinformationssystem – Informationsmodell und Erstellungssystematik, 2002.

145 Schweiker, A.: Offene numerische Steuerungen für prozeßabhängige Bearbeitungen – vereinheitlichte Struktur, Funktionen und Schnittstellen – , 2003.

146 Rempp, B.: Regelungstechnische Untersuchung durchsatzgeregelter Produktionssysteme, 2003.

147 Eppler, C.: Kompensation fremderregter Schwingungen in Antriebssystemen mit Umlaufgetrieben, 2003.

148 Weiner, M.: Wiederverwendungsgerechte Entwicklungssystematik von Feinplanungssoftware für die flexible Fertigung, 2004.

149 McCormac, St.: Lageregelung hydraulischer Manipulatoren unter Einsatz eines Ferraris-Sensors, 2004.

150 Lehner, W.-D.: Regelung von Vorschubachsen unter Verwendung der Relativbeschleunigung, 2005.

151 Lewek, J.: Adaptierbares Informationssystem zur Erstellung baukastenbasierter Fertigungseinrichtungen, 2005.

152 Laible, U.: Aufbau numerischer Steuerungssysteme für sicherheitskritische Anwendungen, 2005.

153 Bürger, Th.: Durchgängige analytische Qualitätssicherung für numerische Steuerungssoftware, 2005.

154 Brinzer, B.: Produktionsregelung für die variantenreiche Serienfertigung, 2005.

155 Heusinger, S.: STEP-NC-basierter Korrekturkreis für die Schlichtbearbeitung von Freiformflächen, 2005.

156 Kirchberger, R.: Verbesserte Auswertung inkrementeller Messsysteme durch schnelle Signal-Vorverarbeitung, 2005.

157 Conrath, M.: Systematische Gestaltung von frequenzadaptierbaren Ultraschall-Werkzeugsystemen zum Einsatz in fertigungstechnischen Prozessen, 2005.

158 Wadehn, W.: Gestaltung von Antriebssystemen für formadaptive Strukturen, 2005.

159 Horber, H.: Fugendetektion bei Lichtbogenschweißprozessen mit robuster Signalverarbeitung für optische Sensoren, 2006.

160 Dreyer, J.: Situative Informationsbereitstellung an Fertigungseinrichtungen Informationsmodell und Erstellungssystematik, 2006.

161 Fritz, S.: Rekonstruktion von Prozesskräften aus Antriebssignalen von Werkzeugmaschinen, 2006.

162 Staudt, S.: Spezifikation und Konformitätstest zur Interoperabilität von automatisierten Produktionsmaschinen, 2006.

163 Reichle, R.: Einsatz von Internet-Werkzeugen und -Diensten in numerischen Steuerungssystemen für Werkzeugmaschinen, 2006.

164 Kaiser, L.: Systematik für das Qualitäts- und Projektmanagement bei der Abwicklung von Unikatprojekten, 2006.

165 Garber, Th.: Nutzung des redundanten Freiheitsgrades von sechsachsigen Parallelkinematik-Maschinen, 2007.

166 Altenburger, R.: Dynamische Eigenschaften und Regelung mechanisch verkoppelter Antriebssysteme, 2007.

167 Korajda, B.: Steuerungstechnische Verfahren zur echtzeitfähigen Kompensation der Fräserabdrängung, 2007.

168 Röck, S.: Echtzeitsimulation von Produktionsanlagen mit realen Steuerungssystemen, 2007.

169 Priese, J.: Verfahren zur durchgehenden dezentralen Planung in Werkstattstrukturen, 2007.

170 Boye, T.: Vorhersage der kinematischen Kalibriergüte von Parallelkinematiken, 2008.

171 Joannides, M.: Ein Beitrag zur volumenorientierten 3D-Objektrekonstruktion aus digitalen Daten, 2009.

172 Pruschek, P.: Verfahren zur anwendungsgerechten Parametrierung der Steuerung und Regelung von Vorschubachsen, 2009.

173 Fritsch, D.: Steuerung selbstorganisierender Multi-Roboter-Systeme für dynamische Sammelaufgaben am Beispiel der Bekämpfung maritimer Ölverschmutzungen, 2009.

174 Schmitz, S.: Industrielle Powerline-Kommunikation für Antriebseinheiten in Werkzeugmaschinen, 2010.

175 Bengel, M.: Workpiece-centered Approach to Reconfiguration in Manufacturing Engineering, 2010.

176 Weimer, T.: Informationsmodell für die durchgängige Datennutzung in Fabrikplanung und -betrieb, 2010.

177 Oglodin, V.: Maschinenübergreifender agentenbasierter Informationsaustausch für die Störungsbeseitigung, 2010.

178 Kramer, C.: Offene Antriebsreglerplattform, 2011.

179 Stotz, M.: Adaptive Segmentierung von Tiefenbildern für die 3-D-Objektlageerkennung auf Basis von kombinierten regelgeometrischen Elementen, 2011.

180 Selig, A.: Informationsmodell zur funktionalen Typisierung von Automatisierungsgeräten, 2011.

181 Meyer, C.: Aufnahme und Nachbearbeitung von Bahnen bei der Programmierung durch Vormachen von Industrierobotern, 2011.

182 Meier, M.: Verfahren zum emulationsgestützten MES-Engineering für die Photovoltaikindustrie, 2011.

183 Walther, M.: Antriebsbasierte Zustandsdiagnose von Vorschubantrieben, 2011.

184 Lechler, A.: Konzeption einer funktional einheitlichen Applikationsschnittstelle für Ethernet-basierte Bussysteme, 2011.

185 Kircher, C.: Selbstadaptierende NC-Steuerung für rekonfigurierbare Werkzeugmaschinen, 2011.

186 Puzik, A.: Genauigkeitssteigerung bei der spanenden Bearbeitung mit Industrierobotern durch Fehlerkompensation mit 3D-Piezo-Ausgleichsaktorik, 2011.

187 Neher, J.: Neuro-Fuzzy-Modellierung zur umfassenden Prozessüberwachung am Beispiel des Ultraschallschweißens von Kunststoffteilen, 2012.

188 Kufner, A.: Automatisierte Erstellung von Maschinenmodellen für die Hardware-in-the-Loop-Simulation von Montagemaschinen, 2012.

189 Sekler, P.: Modellbasierte Berechnung der Systemeigenschaften von Maschinenstrukturen auf der Steuerung, 2012.

190 Huf, A.: Kumulative Lastermittlung aus Antriebsdaten zur Zustandsbewertung von Werkzeugmaschinenkomponenten, 2012.

191 Ledermann, Th.: Partikel-Schwarm-Optimierung zur Objektlageerkennung in Tiefendaten, 2012.

192 Voß, V.: Wiederverwendbare Simulationsmodelle für die domänen- und disziplinübergreifende Produktentwicklung, 2012.