FH Vorarlberg
University of Applied Sciences

SysML Modellierung komplexer Systeme und automatische Erzeugung domänenspezifischer Ressourcen

Masterthesis
zur Erlangung des akademischen Grades

Master of Science in Engineering (MSc)

Fachhochschule Vorarlberg
Mechatronics

Betreut von
Dr. Ralph Hoch

Vorgelegt von
Sebastian Franz
Dornbirn, September 2021

Kurzreferat

Moderne Technische Systeme und ihre Entwicklung werden zunehmend komplexer. Durch eine Vielzahl bei der Entwicklung beteiligter Modelle/Dokumente wird eine konsistente und rückverfolgbare Modellierung erschwert. Model-Based Systems Engineering ist ein Ansatz, welcher diese Problemstellung adressiert und bei der Systementwicklung formale Modelle anstelle von Dokumenten einsetzt/verwendet. Im Gegensatz zu Dokumenten sind Modelle formal definiert, können automatisiert überprüft werden und ermöglichen eine leichte maschinelle Verarbeitung der Informationen. Im Zentrum der Entwicklung steht ein Systemmodell, welches mit einer formalen Modellierungssprache erstellt wurde. Während der Entwicklung entstehende Informationen werden in das Systemmodell integriert, sodass eine konsistente Single Source of Truth entsteht. Aus dem Systemmodell können über automatisierte Modelltransformationen domänenspezifische Ressourcen, wie beispielsweise Digital Twins, generiert werden. Digital Twins und die damit verbundenen Simulationen können einen deutlichen Mehrwert in Bereichen, wie Optimierung, Wartung und Anomalieerkennung liefern. Allerdings ist die manuelle Erzeugung Aufwändig und Fehleranfällig sein, weshalb eine automatische Erzeugung aus den Informationen des Systemmodells eine Erleichterung darstellt.
Diese Arbeit beschäftigt sich mit dieser Problemstellung und der Modellierung komplexer Systeme. Es wird ein Systemmodell mit der Modellierungssprache SysML erstellt, welches durch domänenspezifische Profile erweitert wird um Modellierung/Integration der Domänen zu ermöglichen. Mittels einer Modelltransformation in Acceleo werden Informationen aus dem Systemmodell in ein Simulationsmodell der Simulationssoftware twin, welche zur Simulation von Digital Twins geeignet ist, transformiert. Der in der Arbeit präsentierte Ansatz wird anhand er Modellfabrik des Forschungszentrums Digital Factory Vorarlberg veranschaulicht.

Abstract

Modern technical systems and their development are becoming increasingly complex. A multitude of models/documents involved in the development makes consistent and traceable modelling difficult. Model-Based Systems Engineering is an approach that addresses this problem and uses formal models instead of documents in system development. In contrast to documents, models are formally defined, can be checked automatically and allow easy machine processing of the information. At the centre of the development is a system model created with a formal modelling language. Information generated during development is integrated into the system model to create a consistent single source of truth. Domain-specific resources, such as digital twins, can be generated from the system model via automated model transformations. Digital twins and the associated simulations can provide significant added value in areas such as optimisation, maintenance and anomaly detection. However, manual generation can be cumbersome and error-prone, so automatic generation from system model information can facilitate this.
This thesis deals with this problem and the modelling of complex systems. A system model is created with the modelling language SysML, which is extended by domain-specific profiles to enable modelling/integration of the domains. By means of a model transformation in Acceleo, information from the system model is transformed into a simulation model of the simulation software twin, which is suitable for the simulation of Digital Twins. The approach presented in the paper is illustrated using the model factory of the Digital Factory Vorarlberg research centre.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AutomationML Automation Markup Language

API Application Programming Interface

ATL Atlas Transformation Language

EMF Eclipse Modeling Framework

INCOSE International Council on Systems Engineering

MBSE Model-Based Systems Engineering

MDA Model-Driven Architecture

MES Manufacturing Execution System

MOF Meta Object Facility

MQL Model Query Language

M2M Modell zu Modell

M2T Modell zu Text

OCL Object Constraint Language

OMG Object Management Group

PLM Product Lifecycle Management

QVT Query View Transformation

SCADA Supervisory Control and Data Acquisition

SE Systems Engineering

SM State Machine

STEP STandard for the Exchange of Product model data

SysML OMG Systems Modeling Language

UML Unified Modelling Language

1 Einleitung

In der heutigen Zeit werden technische Systeme zunehmend komplexer. Bei ihrer Entwicklung arbeiten Personen aus verschiedenen Domänen miteinander, wobei jede Domäne ihre eigenen Tools verwendet. Modelle in diesen Tools beschreiben dasselbe System und beinhalten unter Umständen gleiche Informationen, wodurch eine konsistente Modellierung erschwert wird. Zusätzlich unterscheiden sich die Modelle aufgrund ihres Abstraktionslevels voneinander. Während in der Informatik eine Abstraktion durch höhere Programmiersprachen und in weiterer Folge durch Unified Modelling Language (UML) durchgeführt wird, findet eine vergleichbare Methode in den anderen Domänen keine Anwendung. [18] Mit zunehmender Größe solcher Projekte wird das Systems Engineering (SE) dadurch vor Herausforderungen gestellt, welche mit dem bis dato üblichen, auf Dokumenten basierenden Ansatz, immer schwerer umsetzbar sind. Aus diesem Grund wurde ein auf Modellen basierender Ansatz mit den Namen *Model-Based Systems Engineering (MBSE)* entwickelt. MBSE und das damit verbundene Systemmodell kann zur abstrakten Beschreibung technischer Systeme verwendet werden. Das Systemmodell agiert dabei als sogenannte *Single Source of Truth*, wodurch eine konsistente Modellierung gewährleistet wird. Konsistenz deshalb, weil durch die Verwendung einer einheitlichen Modellierung Verweise/Referenzen möglich werden. Für die Modellierung wird eine Modellierungssprache, wie Beispielsweise die *SysML* verwendet. Mit ihr ist es unter anderem möglich, Anforderungen zu modellieren und diese mit Artefakten des Modells zu verknüpfen. Dadurch wird die Rückverfolgbarkeit während der Systementwicklung ermöglicht. Das Systemmodell kann zusätzlich auf seine Richtigkeit überprüft und somit Fehler erkannt und vermieden werden. Durch den technischen Fortschritt und die dadurch immer leistungsfähigeren Computer gewann die Simulation in den letzten Jahrzehnten immer mehr an Bedeutung. Mit Hilfe von Simulationstools können Modelle simuliert werden, um zusätzliche Erkenntnisse über das System zu gewinnen. Simulation ist ein wichtiges Werkzeug für die Vorentwicklung, Entwicklung und Optimierung eines Systems.[34] Modelle und Simulationen können als Grundlage für die Erstellung eines Digital Twins verwendet werden.[10] MBSE und die bei der Digital Twin Technologie benötigte Betriebsdatenerfassung können helfen, das Product Lifecycle Management (PLM) Konzept, welches den gesamten Lebenszyklus eines Produktes/Systems berücksichtigen soll, erstmals ganzheitlich

umzusetzen. Während der Digital Twin Informationen zur Laufzeit eines Systems sammelt, liefert das Systemmodell wichtige Informationen bezüglich der Entstehungsphase.[24]

Diese Arbeit beschäftigt sich mit der Modellierung von komplexen Systemen. Folgende Ziele werden dabei verfolgt:

- Formale Beschreibung des Systems
- Rückverfolgbare Modellierung von Anforderungen
- Integrierung domänenspezifischer Modelle
- Single Source of Truth
- Validierung des Modells
- Anbindung an eine Simulationssoftware

Die Umsetzung wird am Beispiel der Digital Factory und ihrer Modellfabrik realisiert. Das Forschungszentrum Digital Factory Vorarlberg befindet sich in Dornbirn und beschäftigt sich mit aktuellen Themen in den Bereichen Digitale Transformation und Digitalisierung in der Güterproduktion. Dies geschieht in enger Zusammenarbeit mit lokalen Unternehmen. In einer Modellfabrik können wesentliche Aspekte einer modernen Güterproduktion nachgestellt, Lösungen erarbeitet und anschließend veranschaulicht werden. [11] Um das zu ermöglichen verfügt sie über eine Modellfabrik welche aus Robotern, einer Fräsmaschine und einem Intralogisiksystem besteht. Über einen Webshop können Kunden Produkte bestellen. Ein Cloud System übernimmt die Auftragsverwaltung und ein Manufacturing Execution System (MES) die Steuerung des Produktionsablaufes. Ein Supervisory Control and Data Acquisition (SCADA) System kommuniziert über verschiedene Protokolle direkt mit den Maschinen. Die Modellfabrik, welche repräsentativ für die Produktionsstätten der heutigen Zeit ist, besteht somit aus Komponenten aller Domänen der Mechatronik. Eine digitale Abbildung/Modellierung der gesamten Modellfabrik wird aus diesem Grunde sehr komplex und eignet sich deshalb gut als Beispiel für diese Arbeit. Es wird ein Systemmodell mit Hilfe von SysML erstellt. Domänenspezifische Informationen aus einem MCAD Modell sollen in das Systemmodell integriert werden. Ein Simulationsmodell soll in einer Simulationssoftware, welche zur Digital Twin Simulation geeignet ist, erzeugt werden. Grundlage für die Erzeugung des Simulationsmodells sollen dabei die im Systemmodell enthaltenen Daten sein.

Die Arbeit ist folgendermaßen strukturiert: Kapitel 2 befasst sich mit technischen Grundlagen, welche für das Verständnis der Arbeit notwendig sind. In Kapitel 3 werden Arbeiten, welche mit den bearbeiteten Themengebieten in

Verbindung stehen, im Rahmen eines Stand der Technik, bearbeitet. Anschließend wird in Kapitel 4 das geplante Vorgehen beschrieben, welches in Kapitel 5 umgesetzt wird. Das Ergebnis der Arbeit wird in Kapitel 6 präsentiert. Abschließend fasst Kapitel 7 die Arbeit zusammen und beschreibt die noch offenen Punkte.

2 Technisches Hintergrundwissen

Dieses Kapitel gibt eine Einführung in die technischen Grundlagen, in den für die Arbeit relevanten Themengebieten. Eine detailliertere Beschreibung kann den jeweiligen Quellen entnommen werden.

2.1 Systems Engineering

Allgemein handelt es sich bei SE um einen multidisziplinären Ansatz um erfolgreich Systeme zu entwickeln, welche den Bedürfnissen der Stakeholder entsprechen. Es werden dabei technische und wirtschaftliche Aspekte des Systems während seines gesamten Lebenszyklus betrachtet.[15] SE ist ein Ansatz welcher ursprünglich in der Luft- und Raumfahrt, sowie der Verteidigungsindustrie ab Ende der fünfziger Jahre entwickelt wurde. Während des kalten Krieges fand neben dem Wettrüsten auch der Wettlauf ins All statt. Die zu entwickelnden Systeme waren bereits sehr komplex und deren Entwickler standen unter hohem Erfolgsdruck. Mit der Zeit nahm die Komplexität von Systemen in anderen Industriezweigen stark zu, weshalb SE immer mehr an Bedeutung gewonnen hat. 1995 wurde die Organisation International Council on Systems Engineering (INCOSE) gegründet, welche heute über 18.000 Mitglieder aus 35 Ländern hat. Ziel dieser Organisation ist die weltweite Entwicklung und Förderung des Systems Engineering. [16, p. 4]

Aufgabenbereiche des Systems Engineering sind unter anderem

- Projektmanagement
- Risikomanagement
- Requirements Engineering
- Requirements Management
- Systemarchitektur
- Systemverifikation
- Systemvalidierung

- Systemintegration [33, p. 12]

Das V-Modell beschreibt dabei eine V-artige Struktur bei der Systementwicklung, welche von den Anforderungen bis zum fertigen Produkt reicht. Es stammt ursprünglich aus der Softwareentwicklung und wurde von der Richtlinie VDI 2206 zur Entwicklung mechatronischer Systeme empfohlen. In Abbildung 2.1 wird das V-Modell nach VDI 2206 abgebildet. Auf der linken Seite des Modells beginnt, ausgehend von den Anforderungen, der *Systementwurf*. Dabei wird das zu entwickelnde System abstrahiert und disziplinübergreifend entworfen. Dieser Entwurf wird schrittweise verfeinert und im *Domänenspezifischen Entwurf* von den entsprechenden Disziplinen mit bewährten Methoden umgesetzt. Die *Systemintegration* führt die Lösungen der Disziplinen schrittweise zu einem Produkt zusammen. Im Zuge der *Eigenschaftsabsicherung* wird die Validierung und Verifikation des Systems, bei jedem Integrationsschritt durchgeführt.

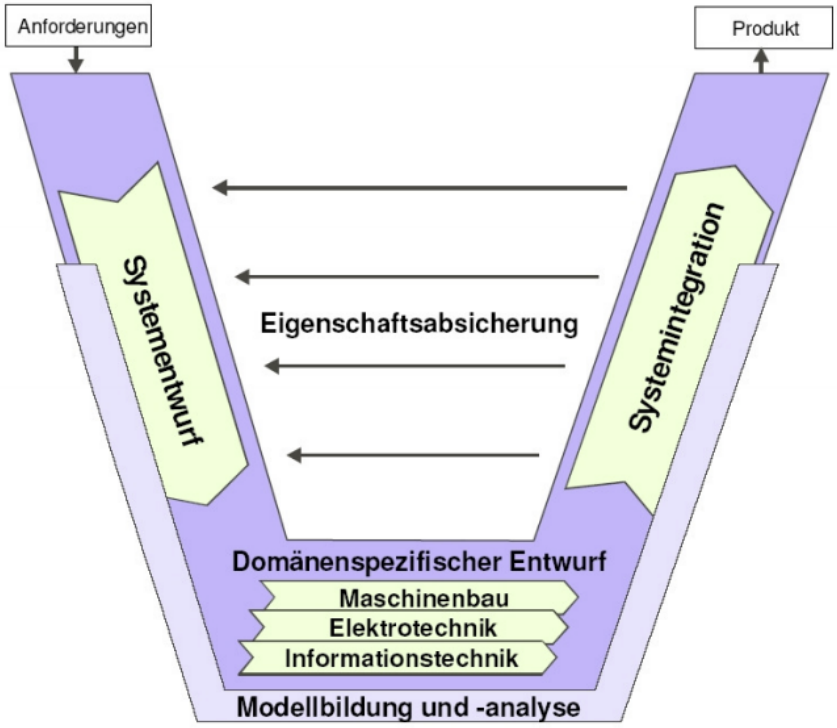


Abbildung 2.1: V-Modell nach VDI 2206

2.2 Model-Based Systems Engineering

Traditionell wurde Systems Engineering mit einem auf Dokumenten basierenden Ansatz umgesetzt. Diese Methode kommt aber gerade bei sehr komplexen Systemen an ihre Grenzen, da Informationen auf vielen Dokumenten verteilt sind und somit die Gefahr von Inkonsistenzen groß ist. Um unter anderem dies zu vermeiden, wurde MBSE entwickelt. Der Begriff MBSE wird von der Organisation INCOSE folgendermaßen definiert:

> "Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases." [17]

Gegenüber dem auf Dokumenten basierenden Ansatz existiert bei MBSE ein Systemmodell. Dieses Modell wird in der Regel mit Hilfe eines Modellierungswerkzeugs erstellt und enthält Informationen bezüglich Systemspezifikation, Design, Analyse und Verifikation. [15, p. 17] Wichtig ist dabei, dass es in sich konsistent sein muss, unterschiedliche Sichten auf die Informationen erlaubt und maschinell auswertbar ist.[33] Zur Erstellung von Systemmodellen werden Modellierungssprachen verwendet. Eine dieser Sprachen ist die SysML.

2.2.1 OMG Systems Modeling Language

Die SysML ist eine graphische Modellierungssprache welche auf der UML basiert und speziell zur Modellierung komplexer multidisziplinärer Systeme entwickelt wurde [9]. Sie wurde 2006 von der Object Management Group (OMG) als Standard angenommen und erschien im September 2007 in der Version 1.0. Der aktuelle Stand ist in Version 1.6 verfügbar und es wird bereits an einer Version 2 gearbeitet. Die SysML ist ein internationales Projekt, weshalb ein Großteil der Literatur in Englisch veröffentlicht wurde. Um Missverständnisse zu vermeiden wird in dieser Arbeit die Englische Benennung von SysML Artefakten verwendet. Die SysML übernimmt einen Teil der UML und erweitert diesen um die Anforderungen des SE abzudecken. Die Beziehung zwischen den beiden Modellierungssprachen wird in Abbildung 2.2 dargestellt. Der Teil welcher von UML in SysML übernommen wurde, wird mit *UML4SysML* bezeichnet. Die Erweiterung wird mit Hilfe eines sogenannten Profils realisiert. Dadurch kann die *UML 2* durch Stereotypes, Tagged Values und Constraints erweitert werden. Da die UML 2 bereits sehr umfangreich ist, wurde jener Teil, welcher für das SE nicht benötigt wird, von der SysML nicht übernommen.

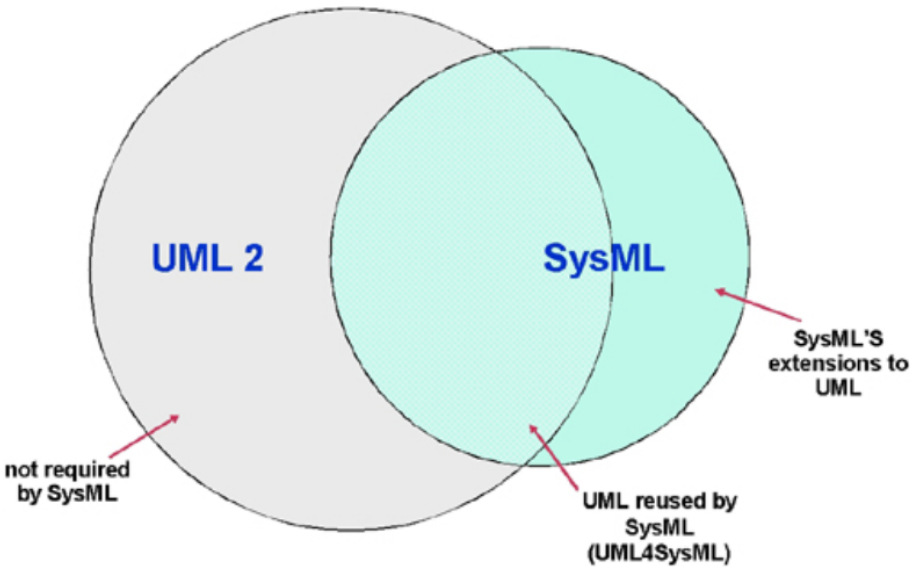


Abbildung 2.2: Beziehung zwischen UML und SysML [35]

Die SysML hat vieles bezüglich Aufbau und Konzept von der UML übernommen. Wie bei der UML gibt es bei der SysML verschiedenen Diagrammtypen, welche in Abbildung 2.3 abgebildet werden. Ein Diagramm bietet eine Sicht auf das System und muss im Vergleich zum Modell nicht vollständig sein. Es gibt in der SysML Diagramme um Anforderungen, Strukturen und Verhalten zu modellieren.

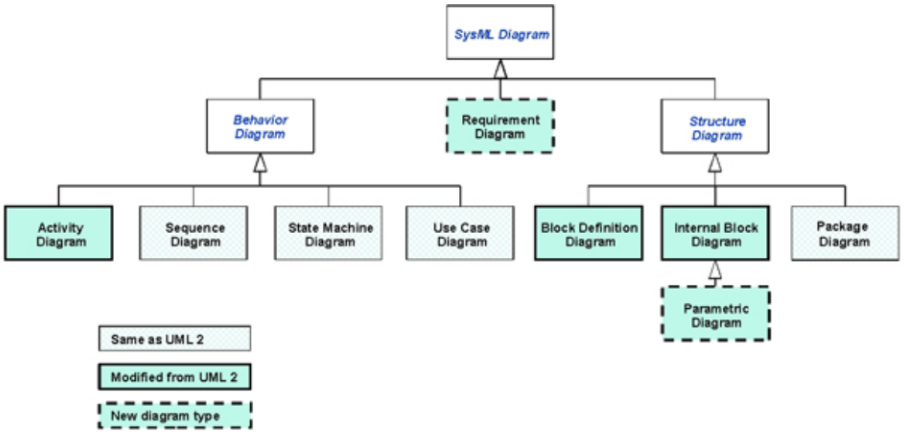


Abbildung 2.3: SysML Diagramme [35]

2.2.1.1 Modellierung von Anforderungen

Die Modellierung von Anforderungen ist in der UML nicht direkt möglich. In der SysML wird mit dem Requirement Diagram ein neuer Diagrammtyp zur Verfügung gestellt. Es ist nun möglich sowohl funktionale als auch nichtfunktionale Anforderungen und Beziehungen zwischen diesen und anderen Modellelementen zu modellieren. In Abbildung 2.4 ist dies anhand eines Beispiels in einem Requirements Diagram dargestellt. Die Anforderungen werden mit Hilfe von *Requirements Blocks* modelliert und besitzen *ID* und *Text*. Mit der *«Satisfy»* Beziehung wird gezeigt, dass ein Modellelement eine Anforderung erfüllen soll.

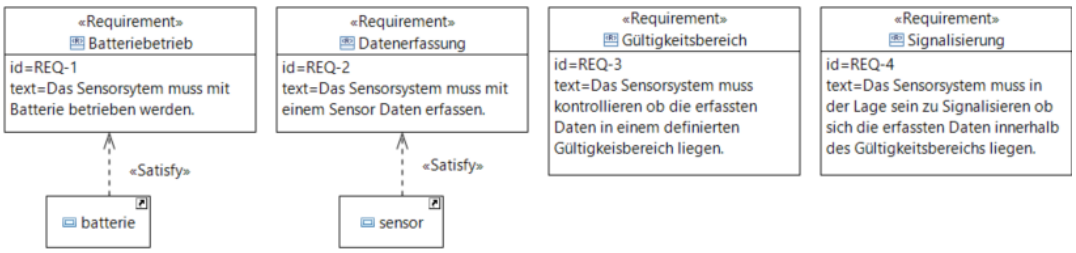


Abbildung 2.4: Beispiel Requirement Diagram

2.2.1.2 Modellierung von Struktur

Ein SysML Modell kann mit Hilfe von Packages strukturiert werden. Es gruppiert Modellelemente und definiert einen Namensraum. Innerhalb eines Packages muss die Benennung der Elemente eindeutig sein. Es ist zudem möglich

Hierarchien von Packages zu erstellen und diese in einem Package Diagram darzustellen. Ähnlich wie die Klasse in UML ist das zentrale Element der SysML der Block. Mit Blocks und verschiedenen Associations kann die Systemstruktur beschrieben und mit den Diagrammen Block Definition Diagram und Internal Block Diagram visualisiert werden. Eine spezielle Art von Block sind sogenannte Constraint Blocks. Mit ihnen können beispielsweise parametrische Beziehungen mit Hilfe von mathematischen Formeln beschrieben werden. Diese können in einen Parametric Diagram, einer speziellen Art des Internal Block Diagrams, dargestellt werden. In Abbildung 2.5 wird ein Block Definition Diagram abgebildet. Es enthält den Block *SensorSystem*, welcher sogenannte Part Properties der Typen *Batterie*, *Sensor*, *Controller* und *RGB LED* besitzt. Im Internal Block Diagram aus Abbildung 2.6 kann die interne Struktur des Block SensorSystem modelliert werden.

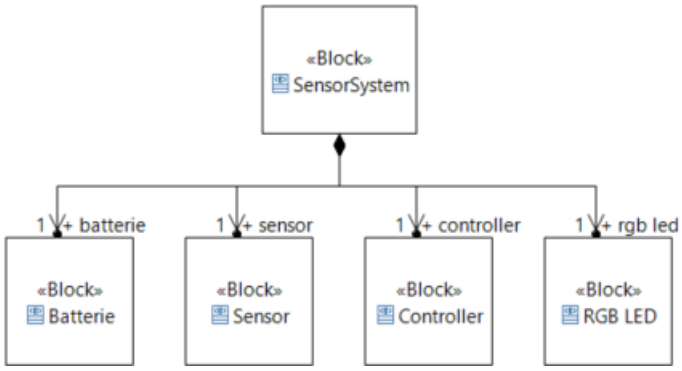


Abbildung 2.5: Beispiel Block Definition Diagram

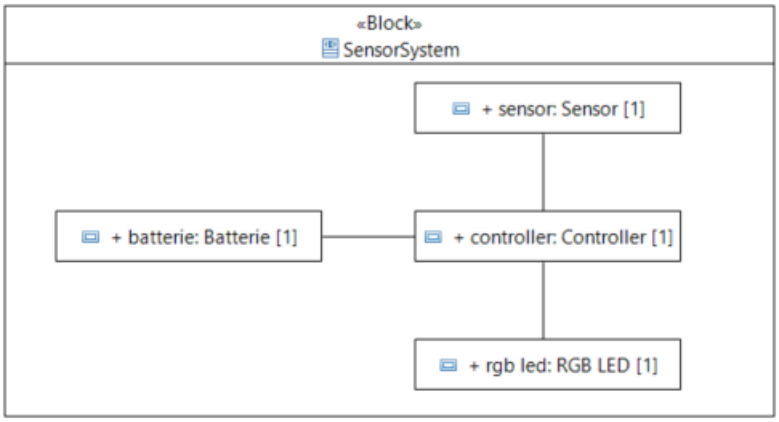


Abbildung 2.6: Beispiel Internal Block Diagram

2.2.1.3 Modellierung von Verhalten

Das Verhalten, beispielsweise jenes des Controllers, kann mit Hilfe der Diagramme Activity Diagram, Sequence Diagram, State Machine Diagram und Use Case Diagram beschrieben werden. Diese Diagramme werden vollständig aus der UML übernommen und das Activity Diagram wird um zusätzliche Eigenschaften erweitert. Erweiterungen sind beispielsweise die Unterstützung der Modellierung kontinuierlicher Systeme, Wahrscheinlichkeiten von Abläufen oder eines Aktivitätsbaumes. In Abbildung 2.7 wird ein mögliches Verhalten des Blocks *Controller* in einem State Machine Diagram dargestellt. Je nachdem ob der Wert des Sensors in einem definierten Wertebereich ist soll in den State *OK* oder *NOK* gewechselt werden.

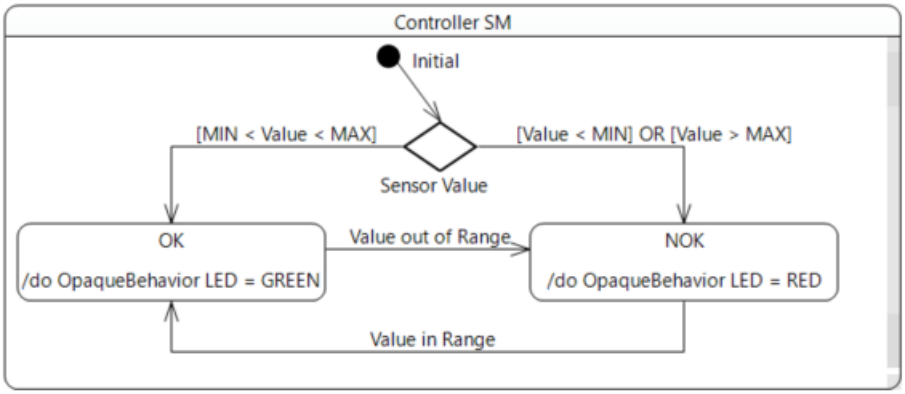
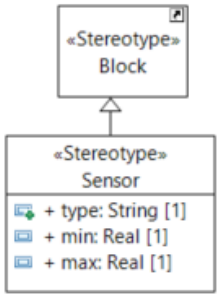


Abbildung 2.7: Beispiel State Machine Diagram

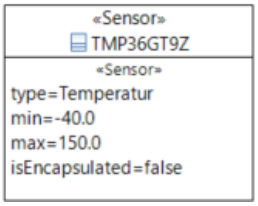
2.2.1.4 Erweiterungsmechanismus

SysML ist eine Modellierungssprache, welche für den allgemeinen Gebrauch entwickelt wurde. Dabei muss sie ein breites Spektrum an Domänen unterstützen. Damit sie sowohl allgemein, als auch domänenspezifisch sein kann, verwendet SysML einen Erweiterungsmechanismus, welcher von der UML übernommen wurde. Er ermöglicht es, die Sprache den jeweiligen Bedürfnissen im Projekt anzupassen, indem neue Modellierungselemente erstellt werden. Diese werden Stereotypen genannt und erweitern bestehende Modellelemente um weitere Eigenschaften und Semantik. Stereotypen können mit Hilfe eines Profils gruppiert, einem Modell zugeordnet und angewendet werden. Zusätzlich unterstützt SysML die Verwendung von Modellbibliotheken. Modellbibliotheken beinhalten wiederverwendbare Modellelemente, während Profile die Modellierungssprache selbst erweitern. Beide Methoden werden verwendet um SysML die domänenspezifische Modellierung zu erleichtern. [15] [33] In Abbildung 2.8 wird die De-

finition und Anwendung des Stereotypes *Sensor* dargestellt. In 2.8a wird der Stereotype Sensor erstellt, welcher den Stereotype *Block* erweitert. Es werden sogenannte Tagged Values *type*, *min* und *max* hinzugefügt. In 2.8b wird der Stereotyp angewendet, und für die Tagged Values Werte gesetzt.



(a) Stereotype Definition



(b) Stereotype Anwendung

Abbildung 2.8: Beispiel Stereotype Sensor

2.2.2 Automation Markup Language

Neben der Erstellung eines Systemmodells, ist der Datenaustausch in einer heterogenen Tool-Landschaft eine wichtige Thematik. Automation Markup Language (AutomationML) ist ein auf XML basierendes, neutrales Datenformat und adressiert den verlustfreien Informationsaustausch zwischen Engineering Tools.[22] Die AutomationML wurde 2006 von der AutomationML Initiative entwickelt. Es wurde kein neues Datenformat konstruiert, sondern bereits bestehende Formate kombiniert. In Abbildung 2.9 wird die Struktur der AutomationML präsentiert. Im Top Level Format CAEX werden Datenformate wie COLLADA oder PLCopen XML zusammengeführt. [5]

2.3 Model-Driven Architecture

Model-Driven Architecture (MDA) ist ein OMG Standard welcher darauf abzielt verschiedene Modelle während der Systementwicklung miteinander in Beziehung zu setzen. Dabei sind Modelle zentraler Bestandteil der Entwicklung und mit Hilfe von Modelltransformationen werden Daten zwischen ihnen ausgetauscht.

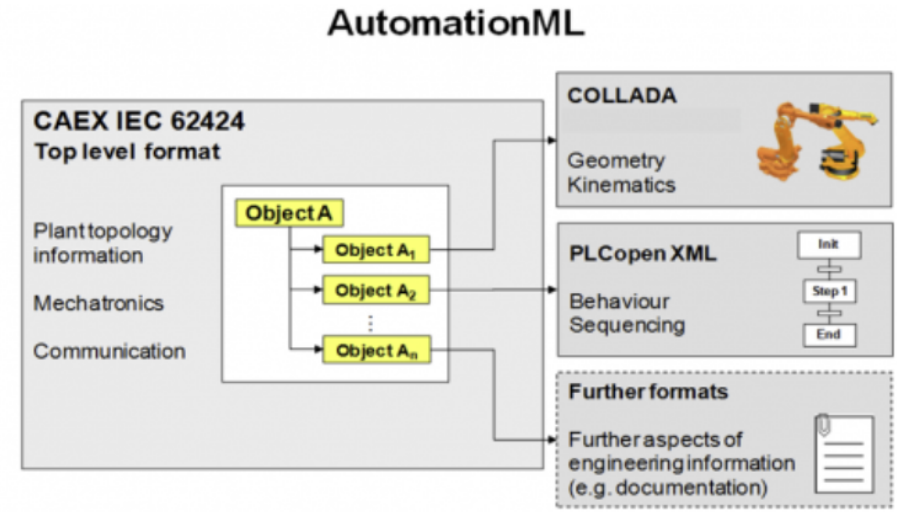


Abbildung 2.9: Struktur der AutomationML

2.3.1 Modelltransformation

Die Modelltransformation ist eine Schlüsselkomponente der modellbasierten Entwicklung. [2] Bei einer Modelltransformation wird aus ein oder mehreren Quellmodellen ein oder mehrere Zielmodelle erzeugt. Man unterscheidet dabei zwischen Modell zu Modell (M2M) und Modell zu Text (M2T) Transformationen. Während das Ziel bei einer M2M Transformation wieder ein Modell ist, wird bei einer M2T Transformation Text/Code generiert. Eine M2M Transformation besteht typischerweise aus Transformationsregeln und einer Transformation Engine. Eine Tranformationsregel beinhaltet Informationen wann sie angewendet wird und wie Elemente im Zielmodell erzeugt werden sollen. Die Transformation Engine führt die Regeln meist in zwei Schritten aus. Als erstes wird in den Quellmodellen nach Elementen gesucht, welche zu einer Ausführung der Regel führen. Anschließend werden für diese die entsprechenden Elemente der Zielmodelle erzeugt. Die Modellstruktur der beteiligten Modelle muss in Form von Metamodellen vorliegen. Die darin enthaltenen Informationen werden für die Erstellung und Ausführung der Transformation benötigt.[1] Eine M2M Transformation kann Beispielsweise mit der Atlas Transformation Language (ATL) (siehe Unterunterabschnitt 2.3.1.2) durchgeführt werden. Bei einer M2T Transformation wird aus Quellmodellen Text, beziehungsweise Code, generiert. Eine Möglichkeit für die Umsetzung der Transformation ist die Verwendung von Templates. Dabei werden Artefakte der Quellmodelle in bereits vorgefertigte Code-Fragmente eingesetzt. Sprachen welche zur M2T Transformation mit Hilfe von Templates verwendet werden können sind Xtend (siehe

2.3.1.3) oder Acceleo (siehe 2.3.1.4).

2.3.1.1 Metamodell

Metamodelle können zur Definition von Modellierungssprachen verwendet werden und sind wiederum Modelle. Die Modellierungssprache mit welchem dieses Metamodell erstellt wurde, kann wiederum mit Hilfe eines Meta-Metamodells definiert werden. Meta Object Facility (MOF) ist eine Meta-Metamodellsprache, welche im Umfeld von SysML sehr verbreitet ist. Ecore ist ebenfalls eine Sprache zur Erzeugung von Metamodellen und der MOF sehr ähnlich. Sie kann als Implementierung einer Teilmenge, welche Essential MOF genannt wird, bezeichnet werden. In Abbildung 2.10 wird das Ecore Modell *Families.ecore* abgebildet. Es enthält die Elemente *Family* und *Member* vom Typ *EClass*. Mit dem EAttribute Elementen *lastName* und *firstName* wird modelliert, dass jede Family einen Nachnamen und jeder Member einen Vornamen besitzt. Durch die EReference Elemente *father*, *mother*, *sons* und *daughters* wird eine Verbindung zwischen den EClass Elementen Family und Member hergestellt.

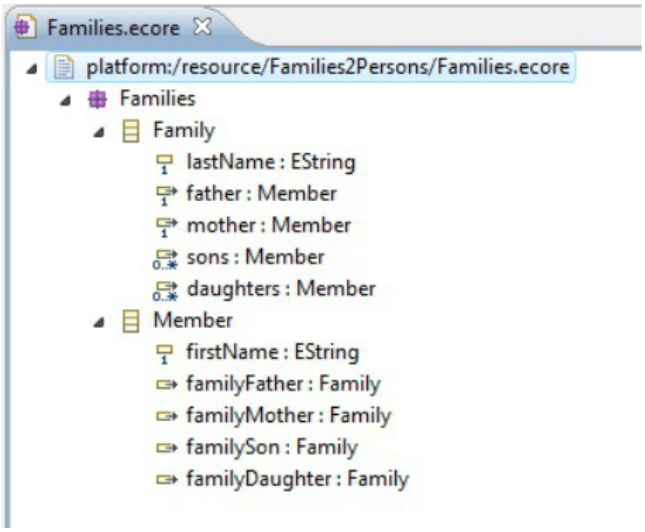


Abbildung 2.10: Beispiel Ecore Metamodell [4]

2.3.1.2 Atlas Transformation Language

ATL ist eine Programmiersprache welche aus einer Anzahl von Quellmodellen ein Set aus Zielmodellen erzeugt. Sie vereint dabei Konzepte imperativer und deklarativer Programmierung, wobei ein deklarativer Transformationsstil empfohlen wird. Eine M2M Transformation wird in einem ATL Module beschrieben,

welches aus einem Header, Imports von Libraries und beliebig vielen Helpers und Rules besteht. Zusätzlich gibt es sogenannte ATL Queries und ATL Libraries. Mit ATL Queries ist es möglich aus Quellmodellen primitive Datentypen wie Strings oder Integer zu berechnen. Mit Hilfe von Libraries kann ATL Code mehrfach verwendet werden.[3]

2.3.1.3 Xtend

Xtend ist eine statisch getypte Programmiersprache welche ihre Wurzeln in der Programmiersprache Java hat. Xtend interagiert nahtlos mit Java, ist jedoch um einiges prägnanter, lesbarer und ausdrucksvoller. Obwohl es sich um eine allgemeine Programmiersprache handelt, eignet sie sich auch für Modelltransformationen. Mit Hilfe von sogenannten Template Expressions können beispielsweise M2T Transformationen erstellt werden.[36]

2.3.1.4 Acceleo

Acceleo ist eine Template Basierte Open Source Technologie zur Codegenerierung welche von der Eclipse-Foundation zur Verfügung gestellt wird. Die Sprache die dabei verwendet wird ist eine Implementierung des MOFM2T Standards. Acceleo basiert auf dem Eclipse Modeling Framework (EMF) Standard und kann deshalb mit anderen Tools dieses Standards verwendet werden. Eine Transformation mit Acceleo besteht im allgemeinen aus sogenannten *Modules*, *Templates* und *Queries*. Ein Module ist ein *.mtl-File*, welches Templates und Queries beinhaltet. Eine Transformation kann auf mehrere Files aufgeteilt werden. In Templates können statische Ausdrücke (Text) und Acceleo Ausdrücke verwendet werden. Acceleo Ausdrücke können verwendet werden um Informationen aus dem Eingangsmodell zu extrahieren und zusammen mit Text in Files zu speichern. Für die Abfragen werden *Object Constraint Language (OCL)-Ausdrücke* verwendet, welche direkt im Template verwendet oder in Queries ausgelagert werden können. Mit Hilfe von *for-Schleifen* und *if-Ausdrücken* kann durch die Elemente des Eingangsmodells navigiert werden. [14]

2.4 Product Lifecycle Management

PLM ist ein Ansatz zur unternehmensweiten Verwaltung eines Produktes über seinen gesamten Lebenszyklus und soll dabei helfen Produktdaten Konsistent zu halten. Die Ziele von PLM überschneiden sich mit denen des SE zu einem großen Teil, weshalb die Ansätze kombiniert werden können. [33] [28]

2.5 Digital Twin

Ein Digital Twin ist ein Modell einer physikalischen Anlage, welches mit Hilfe von Algorithmen mit Prozessdaten versorgt wird. Typische Anwendungsbereiche für einen Digital Twin sind Prozessoptimierung, Predictive Maintenance oder Anomalienerkennung inklusive Fehlerisolierung. [23] Ein Digital Twin entsteht durch die sinnvolle Verknüpfung einer Instanz des Digital Masters mit dem Digital Shadow. Der Digital Master besteht aus Modellen die während der Entwicklungsphase entstehen. Der Digital Shadow besteht aus Prozessdaten der physischen Anlage. [30] Der resultierende Digital Twin wird dabei laufend mit Prozessdaten versorgt. Diese Daten können zur Simulation verwendet werden. Das Resultat einer Simulation könnte eine Parameteränderung bei der Anlage zur Folge haben (Prozessoptimierung). Ein anderes Resultat könnte eine Prognose bezüglich künftiger Wartungszeitpunkte sein (Predictive Maintenance). Außerdem kann mit Hilfe von Algorithmen in den Daten nach Anomalien gesucht werden.

3 State of the Art

Shah et al. [29] präsentieren in ihrer Arbeit ein "Model Integration Framework" für multidisziplinäre Modellierung. Dabei werden die involvierten Disziplinen in Metamodellen mit Hilfe des MOF Standards formal definiert. SysML wird mit Hilfe von Profilen erweitert, um diziplinspezifische Modellierung zu ermöglichen. Mit Hilfe von Query View Transformation (QVT) wird ein Mapping zwischen Metamodellen und Profilen erstellt. Tool spezifische API-Calls in den Transformationen vervollständigen die integration zwischen den Disziplinen. Der Ansatz wird anhand des Beispiels von EPLAN uns SysML Modellen veranschaulicht. In Abbildung Abbildung 3.1 wird der Integrationsansatz dargestellt. Mit Hilfe eines SysML Profils und der darin enthaltenen Stereotypen wird ein domänenspezifisches (EPLAN) Modell in SysML erstellt. Dieses wird mit Hilfe des definierten Metamodells in ein vorläufiges Domänenmodell transformiert. Mit Hilfe von Funktionsaufrufen der internen Application Programming Interface (API) des EPLAN Tools wird dieses Modell in ein EPLAN Modell umgewandelt. Wie in [29] wird auch in dieser Arbeit aus einem SysML Modell ein Zielmodell erzeugt. Es wird ebenfalls eine Modelltransformation und eine API des Zielmodell-Tools verwendet. Jedoch wird mit der Transformation direkt Code generiert, welcher von der API verwendet werden kann, um das Zielmodell zu erzeugen.

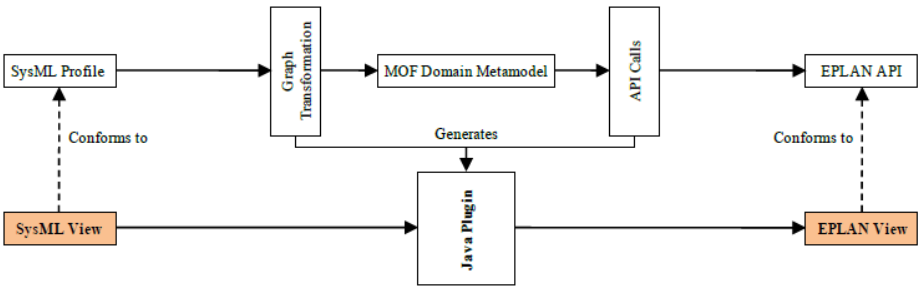


Abbildung 3.1: Multi-View Modeling Method, SysML-view und EPLAN-view [29]

Qamar [27] präsentiert in seiner Arbeit einen Ansatz um domänenspezifische

Modelle mit einem SysML Modell zu verknüpfen. SysML Profile werden für die domänenspezifische Modellierung in SysML erstellt. Mittels Verwendung des EMF wird die Modellintegration realisiert, indem Metamodelle sowohl für die Profile als auch die Domänen in Ecore erstellt, und Transformationen zwischen ihnen definiert werden. Die Anbindung an die spezifischen Tools wird mit sogenannten Tool Adaptern realisiert. In Abbildung 3.2 wird die Infrastruktur für die Integration einer Domäne abgebildet. Der Ansatz wird anhand des Beispiels von MCAD in der Arbeit ausgeführt. Für die Transformation werden ATL und Model Query Language (MQL) verwendet. Für die Anbindung an MCAD wird ein Tool Adapter in Visual Basic geschrieben, welcher in der Lage ist aus einem parametrisierten MCAD Modell ein XML File zu erzeugen, welches der Struktur des Metamodells entspricht und umgekehrt. Für die MCAD Modellierung der Digital Factory in SysML wird ein Profil, wie das in [27], verwendet werden. Eine Anbindung an ein MCAD Tool wird nicht Teil der Arbeit sein. Stattdessen wird versucht die Informationen in ein zentrales Systemmodell zu integrieren und in weiterer Folge die Generierung eines Simulationsmodells zu realisieren.

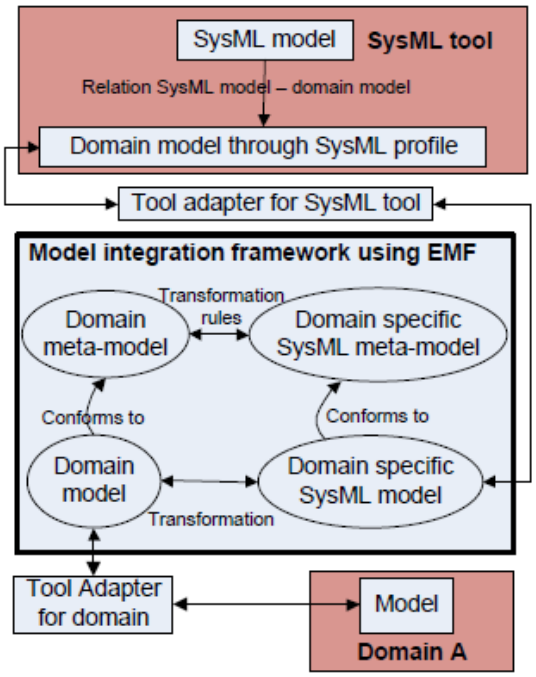


Abbildung 3.2: Integrationsinfrastruktur einer Domäne [27]

Thramboulidis [32] präsentiert in seiner Arbeit das 3+1 SysML view-model, dessen Aufbau in Abbildung 3.3 abgebildet ist. Es gibt ein zentrales Modell (*MTS-View*), welches das gesamte mechatronische System beschreibt. Zusätzlich gibt

es noch drei disziplinspezifische Modelle (*s-View*, *e-View* und *m-View*), welche das System aus ihrer Perspektive beschreiben und mit dem zentralen Modell synchronisiert werden. Ein SysML Profil namens SysML4MIM wird dabei verwendet, um Anforderungen und Architektur des mechatronischen Systems und dessen Komponenten zu beschreiben. Die Modellierung wird in das im Jahre 2005 vom selben Autor präsentierte Model Integration Mechatronics Paradigma [31] eingebettet.

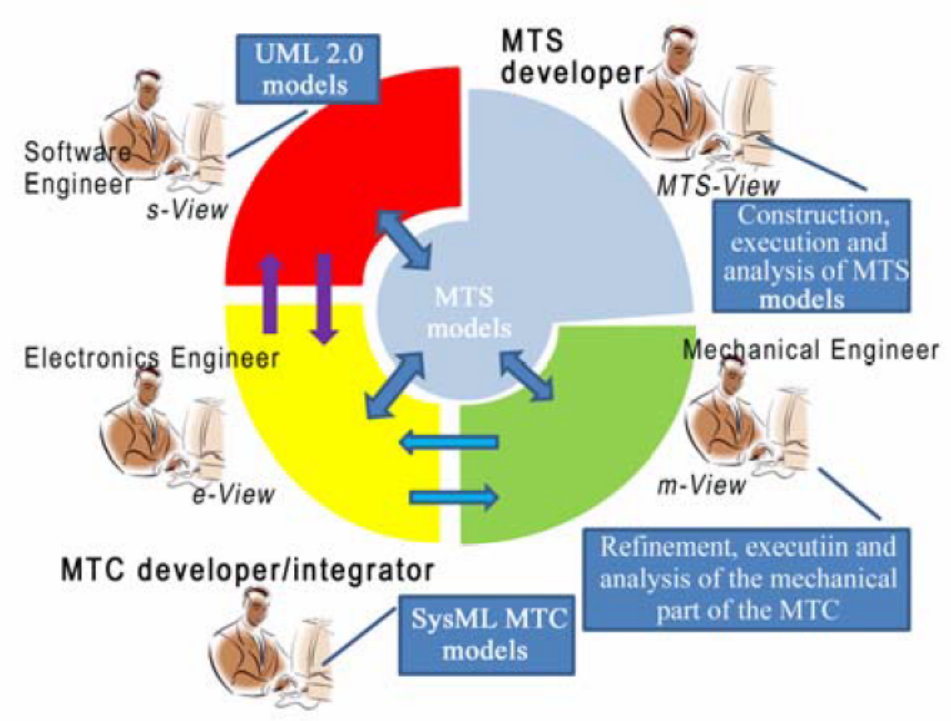


Abbildung 3.3: SysML4MIM Profile [32]

Bassi et al. [7] verwenden in ihrem Ansatz eine Hierarchie von Modellen, welche das System auf verschiedenen Abstraktionsebenen beschreiben. Diese Ebenen sollen über Mappings miteinander verknüpft werden. Die höchste Abstraktionsebene wird mit Hilfe von SysML beschrieben, während die unterste Ebene bestmögliche disziplinspezifische Tools verwendet um Subsysteme so genau wie möglich zu modellieren. Dies wird mit einem iterativen Designprozess und Validierungsregeln kombiniert und anhand des Beispiels einer Verpackungsmaschine präsentiert. Ähnlich wie in [7] wird die Digital Factory in unterschiedlichen Abstraktionsebenen modelliert. Die niedrigste Ebene ist dabei das MCAD Modell, die mittlere Ebene das Domänenmodell in SysML und die höchste das System-

modell in SysML.

Kernschmidt und Vogel-Heuser [20] beschreiben einen Modellierungsansatz zur Darstellung und Analyse von Einflüssen, welche Änderung in mechatronischen Produktionsanlagen mit sich bringen. Eine solche Änderung könnte beispielsweise ein neuer Sensor, welcher mit einem bestehenden getauscht wird, sein. Der Ansatz wird mit Hilfe der SysML umgesetzt und trägt den Namen SysML4Mechatronics. Ein gemeinsames Model der Struktur des Systems wird in drei disziplinspezifischen Ansichten dargestellt. Dabei enthält jede Ansicht nur die entsprechend notwendigen Informationen. Interaktionspunkte der jeweiligen Komponenten werden mit Hilfe von Ports modelliert. Internal Block Diagrams werden verwendet, um Komponenten und die entsprechenden Interaktionen in den jeweiligen Ansichten darzustellen. Der Ansatz soll die interdisziplinäre Analyse von Änderungseinflüssen, welche manuell durchgeführt wird, erleichtern.

Barbieri et al. [6] präsentieren in ihrer Arbeit einen auf SysML basierenden Prozess um mechatronische Systeme auf einer höheren Abstraktionsebene zu beschreiben. Für den Design Prozess wird das V-Modell angewendet, welches sich in Breakdown Phase (linke Seite) und Integration Phase (rechte Seite) unterteilen lässt. Ein Requirements Profile wird erstellt, um Anforderungen durch alle Abstraktionsebenen rückverfolgbar modellieren zu können. Für die Modellierung der Struktur wird das Profile SysML4Mechatronics verwendet, welches unter anderem in [19] beschrieben wird. Das Verhalten wird mit Hilfe von Activity Diagrams und State Machines modelliert. Ein Vorteil des beschriebenen Prozesses ist die Wiederverwendbarkeit von Modellen und die Rückverfolgbarkeit der Modellinformationen. Wie in [6] wird sich auch in dieser Arbeit während des Design Prozesses an das V-Modell gehalten. Es wird jedoch für die Modellierung im Systemmodell auf Profile verzichtet. Diese könnten dem präsentierten Ansatz bei Bedarf jedoch hinzugefügt werden. Eine rückverfolgbare Modellierung von Anforderungen auf unterschiedlichen Abstraktionsebenen wird durch die «deriveReq»Beziehung realisiert.

Berardinelli et al. [8] beschreiben einen Ansatz, welcher SysML und AutomationML verknüpft. Die Arbeit veranschaulicht anhand des Beispiels eines kleinen, aber repräsentativen Produktionssystems die Modellierung mit Hilfe von SysML und AutomationML. Es wird dabei der Fokus auf die Modellierung der Struktur (CAEX in AutomationML und Blockdiagramme in SysML) gelegt. Der Artikel untersucht Gemeinsamkeiten und Unterschiede der beiden Standards um Interoperabilität herzustellen. Es wird ein AML4SysML Profile erstellt, welches CAEX und SysML durch Mapping miteinander verknüpft. Außerdem wird die

Transformation zwischen den beiden Sprachen mit Hilfe von ATL präsentiert. Die Architektur zur Verknüpfung der beiden Sprachen wird in Abbildung 3.4 gezeigt. Eine Anbindung der unterschiedlichen Domänen an ein SysML Modell mit Hilfe von AutomationML wäre auch für diese Arbeit eine Möglichkeit. Dafür müsste das Tool der jeweiligen Domänen in der Lage sein, das Modell im AutomationML Format zu exportieren. Ein Mapping zwischen AutomationML und SysML könnte wie in [8] umgesetzt werden. Eine solche Anbindung wird jedoch als zu Aufwändig eingeschätzt und ist deshalb nicht Teil der Arbeit.

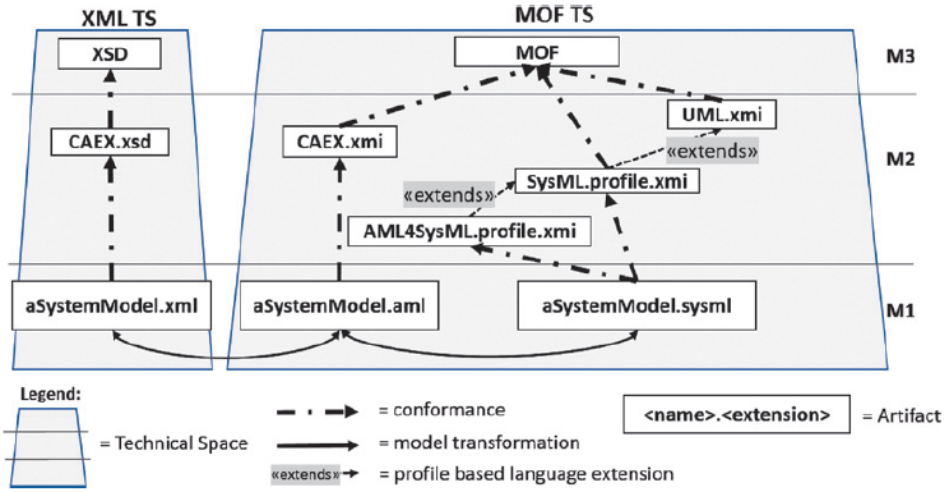


Abbildung 3.4: Verknüpfung von AutomationML und SysML[8]

Die Autoren Li et al. präsentieren in ihrer Arbeit [21] einen Ansatz wie multidisziplinäre Systeme im Rahmen von MBSE effizient modelliert werden können. Es wird dafür das SysML Profil SysML4Mechatronics verwendet. Mit Hilfe dieses Profils wird ein Systemmodell namens „SysML4Mechatronics plant model“ erstellt. Als Modell Datenbank und zur Versionsverwaltung kommt eine PLM Software zum Einsatz. Der Datenaustausch zwischen disziplinspezifischen Tools, SysML und PLM funktioniert über Files im AutomationML Format. Der Ablauf wird in Abbildung 3.5 abgebildet.

Eigner et al. [12] präsentieren in ihrer Arbeit unter anderem ein Forschungsprojekt namens „mecPro^2“. Ziel des Projekts ist es, die Entwicklung cybertronischer Systeme, durch die Verwendung von MBSE, effizienter zu gestalten. Eine detailierte Beschreibung des Projekts wurde in [13] veröffentlicht.

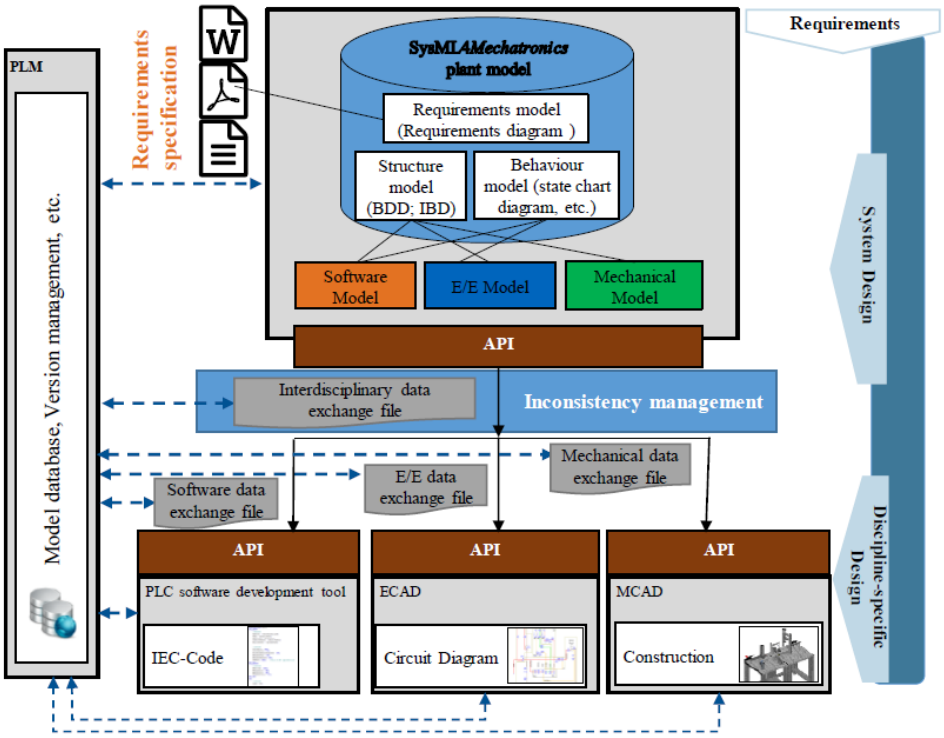


Abbildung 3.5: SysML4Mechatronics Plant Model [21]

Die Arbeiten [21] und [12] zeigen, wie sich PLM und MBSE kombinieren lassen. Es werden komplexe Strukturen zur Integration domänenspezifischer Modelle präsentiert. Die zur Verfügung stehenden Ressourcen lassen eine solch aufwändige Art der Modellierung nicht zu, zumal ein wesentliches Ziel der Arbeit die Anbindung an ein Simulationsmodell ist.

4 Modellierungskonzept für komplexe Systeme

In diesem Kapitel wird das geplante Konzept mit all seinen Elementen beschrieben. Zentrales Element ist ein Modell, welches in der Modellierungssprache SysML verfasst wird. Es soll zur formalen Beschreibung des Systems dienen und als eine Art Single Source of Truth Informationen aus domänenspezifischen Modellen integrieren. Die im Modell enthaltenen Informationen sollen zur Generierung eines Simulationsmodells, zur Simulation Digitaler Zwillinge, verwendet werden.

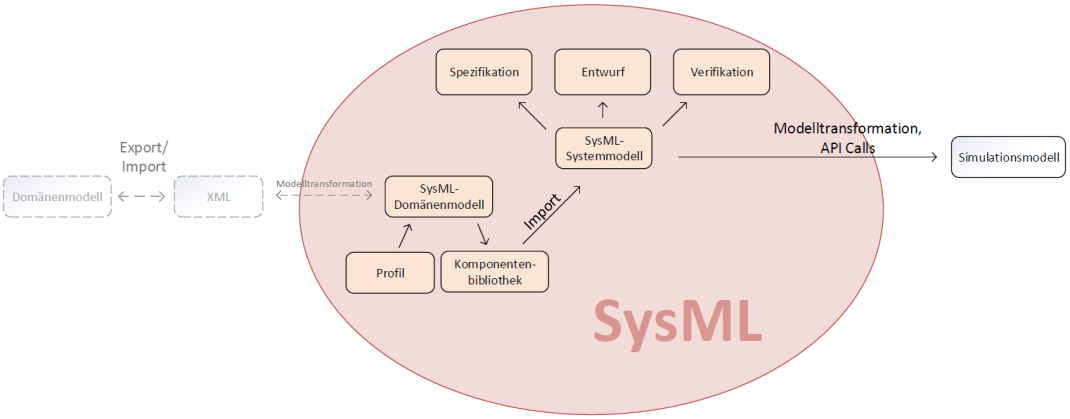


Abbildung 4.1: Modellierungskonzept

In Abbildung 4.1 wird das geplante Vorgehen anhand einer Grafik präsentiert. Detaillierte Modelle aus den jeweiligen Domänen werden in ein neutrales Datenformat exportiert. Über eine Modelltransformation kann die enthaltene Information in ein *SysML-Domänenmodell* transformiert werden. Die beschriebene Anbindung an das SysML-Domänenmodell wird nicht Teil der Arbeit sein und ist deshalb in der Grafik grau dargestellt. Im SysML-Domänenmodell wird ein *Profil* verwendet, um die jeweilige Domäne zu modellieren. Die erzeugten Artefakte werden in einer *Komponentenbibliothek* gespeichert und in ein *SysML-Systemmodell* importiert. Dieses beinhaltet Informationen bezüglich *Spezifikation*, *Entwurf* und *Verifikation* des Systems. Bibliothekskomponenten werden

mit Artefakten des Systementwurfs verknüpft. Mittels Modelltransformation und einer API der Simulationsumgebung werden aus dem SysML-Systemmodell Teile eines *Simulationsmodells* generiert.

4.1 SysML-Systemmodell

Ein wesentlicher Teil der Arbeit ist die Systemmodellierung mit der SysML. Ein Systemmodell beinhaltet Informationen bezüglich der Systemspezifikation, Systementwurf und Verifikation. Die Systemspezifikation beinhaltet Anforderungen (*Requirements*) an das Gesamtsystem, seiner Subsysteme und deren Komponenten. Diese dienen als Basis für den Systementwurf, welcher Struktur und Verhalten beinhaltet. Die Systemstruktur wird mit Hilfe von Blöcken (*Blocks*) und deren Beziehungen zueinander aufgebaut. Das Systemverhalten wird unter anderem durch SMs modelliert. Anforderungen werden über spezielle Beziehungen, wie *«verify»* oder *«satisfy»*, mit anderen Artefakten des Modells verknüpft und dadurch rückverfolgbar modelliert. SysML Modelle werden mit steigender Anzahl von Modellelementen zunehmend unübersichtlicher. Eine Strukturierung mit Hilfe von Paketen (*Packages*) ist deshalb in jedem Fall empfehlenswert. In dieser Arbeit wird die Top-Level-Struktur des Systemmodells am Beispiel des Vortrags von Peleska [26] gewählt. Sie wird für den Großteil multidisziplinärer Systeme als passend beschrieben und enthält folgende Packages:

- Requirements
  Enthält alle Anforderungen an das zu entwickelnde System.
- Context
  Beschreibt das zu entwickelnde System in seiner Betriebsumgebung. Dabei werden die Schnittstellen zur Umgebung modelliert.
- DigitalFactory
  Beschreibt das zu entwickelnde System mit seinen strukturellen, funktionalen sowie nicht-funktionalen Modellelementen. Es trägt den Namen des zu entwickelnden Systems.
- SystemInterfaces
  Enthält Informationen bezüglich der Schnittstellen des zu entwickelnden Systems
- ProjectTypes
  Enthält die im Projekt verwendeten Datentypen, welche mit Hilfe von Enumerationen und Blöcken erstellt wurden.

- PhysicalUnits
  Enthält Physikalische Einheiten welche bei der Systemmodellierung benötigt werden.
- Test
  Enthält Test Design und System Test Submodelle
- ImportedPackages
  Enthält alle importierten Packages

Diese Arbeit beschäftigt sich hauptsächlich mit dem Inhalt der Packages *Requirements* , *DigitalFactory* und *Imported Packages*.

4.1.1 Systemspezifikation

Gemäß des V-Modells aus Abbildung 2.1 sind Anforderungen der Ausgangspunkt für die Systementwicklung. Sie werden im Package *Requirements* durch Blöcke mit dem Stereotyp *«Requirement»*, wie in Abbildung 4.2, modelliert. Sie beinhalten eine eindeutige ID und Text, welcher die Anforderung an das System beschreibt.

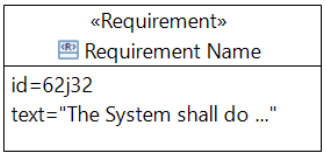


Abbildung 4.2: «Requirement» Block [25]

Durch die sogenannte *Containment* Beziehung können Hierarchien von Anforderungen, wie in Abbildung 4.3, erstellt werden. Es gibt unterschiedliche Ebenen von Anforderungen, welche sich in ihrem Detaillierungsgrad voneinander unterscheiden.

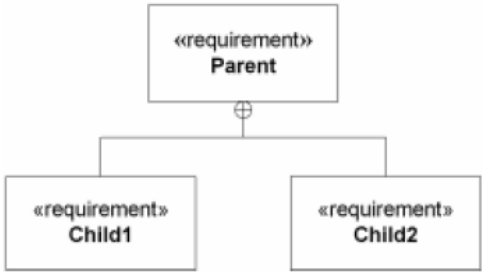


Abbildung 4.3: Containment Beziehung [25]

Über die *«DeriveReqt»* Beziehung können Anforderungen von unterschiedlichen Abstraktionsebenen miteinander verbunden werden (Abbildung 4.4). Während des SE-Prozesses werden Anforderungen auf diese Art schrittweise verfeinert, bis sie nicht mehr weiter aufgeteilt werden können und als Atomar gelten.

Abbildung 4.4: «deriveReqt» Beziehung [25]

In der SysML können mit Hilfe der *«satisfy»* Beziehung Anforderungen mit Systemkomponenten verknüpft werden (Abbildung 4.5). Somit kann beispielsweise kontrolliert werden, ob alle Anforderungen im Systementwurf berücksichtigt wurden.

Abbildung 4.5: «satisfy» Beziehung [25]

4.1.2 Systementwurf

Der Systementwurf wird auf Basis der Systemspezifikation erstellt. Er wird im Package *DigitalFactory* modelliert und beinhaltet sowohl die Struktur als auch das Verhalten des Systems und dessen Komponenten. Mit Hilfe der *«allocate»* Beziehung aus Abbildung 4.6 kann ein Mapping zwischen Struktur und Verhalten hergestellt werden. Sie ist eine Erweiterung des UML Typs *Abstraction*.

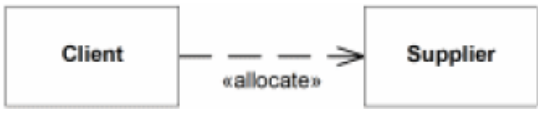


Abbildung 4.6: «allocate» Beziehung [25]

4.1.2.1 Struktur

Die Systemstruktur wird mit Hilfe von Blöcken und der sogenannten *„Composite Association“* aufgebaut. Dadurch wird eine Teil/Ganzes-Beziehung zwi-

schen den Blöcken hergestellt. In Abbildung 4.7 wird dies anhand der Blöcke *Block1* und *Block2* in einem *Block Definition Diagram* dargestellt. Durch die verwendete Assoziation wird definiert das, Block1 beliebig viele sogenannte *Part Properties* vom Typ Block2 besitzen kann. Dieses Konzept wird für die Strukturmodellierung verwendet. Basis für die Strukturmodellierung sind die im Package *Requirements* modellierten Anforderungen. Die Struktur wird dabei schrittweise verfeinert, bis schlussendlich alle Anforderungen berücksichtigt und die benötigten Komponenten identifiziert wurden. Ein Vorteil einer solchen schrittweisen Abstraktion ist, dass Änderungen auf niedrigeren Abstraktionsebenen meist keinen Einfluss auf höhere Ebenen des Modells haben.

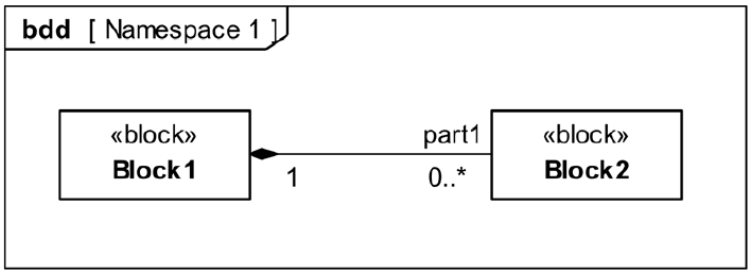


Abbildung 4.7: Strukturmodellierung mit Blöcken und Assoziationen [25]

4.1.2.2 Verhalten

Verhaltensmodelle können mit den in Abbildung 2.3 abgebildeten Verhaltensdiagrammen modelliert werden. In dieser Arbeit wird Verhalten von Systemkomponenten mit Hilfe von SMs modelliert. In einer SM gibt es verschiedene Zuständen (*States*), in welchen die Komponente ein bestimmtes Verhalten aufweist. Die Komponente befindet sich immer in einem dieser Zustände. Sogenannte *Change Events* lösen Transitionen (*Transitions*) zwischen den Zuständen aus.

4.1.3 Verifikation

Ob der Systementwurf die Anforderungen der Sytemspezifikation wirklich erfüllt, kann durch Tests überprüft werden. Diese können in SysML mit dem Stereotyp *«testCase»* definiert und mit der *«verify»* Beziehung mit Anforderungen verknüpft werden (Abbildung 4.8). Der Testablauf kann durch SysML Diagramme beschrieben werden. Die Tests und deren Abläufe können im Package *Test* modelliert werden.

Abbildung 4.8: «verify» Beziehung [25]

4.2 SysML-Domänenmodell

Die Modellfabrik der Digital Factory Vorarlberg besteht aus Mechanik-, Elektronik- und Softwarekomponenten. Modelle der jeweiligen Domänen können am besten mit Hilfe eigener Modellierungswerkzeuge erstellt werden. So wird beispielsweise die physikalische Struktur der Modellfabrik als Baugruppe in einer CAD Software modelliert. Die elektrischen Komponenten und deren Verschaltung können in EPLAN und Prozessabläufe mit Hilfe von UML Diagrammen modelliert werden. Für die domänenspezifische Modellierung mit der SysML kann die Sprache durch Profile (*Profiles*) erweitert werden. Eine Domäne wird durch die Verwendung des jeweiligen Profils in einem SysML-Domänenmodell modelliert. Ein Integrationsprozess verknüpft das Modell eines domänenspezifischen Tools mit dem SysML-Domänenmodell. Der Integrationsprozess ist nicht Teil der Arbeit. Die Arbeiten [29] und [27] liefern Beispiele für Integrationsprozesse dieser Art. Für diese Arbeit wird von einem SysML-Domänenmodell, in welchem der physische Aufbau der Modellfabrik aus einem MCAD Modell händisch übernommen wurde, ausgegangen. Die Modellierung wird durch ein Profil, vergleichbar mit jenem aus [27], ermöglicht.

4.2.1 Anbindung an das SysML-Systemmodell

Packages des SysML-Domänenmodells können durch den Stereotyp *«ModelLibrary»* (aus UML::StandardProfile) als Modellbibliothek gespeichert und in das SysML-Systemmodell importiert werden. Durch die Verwendung von Bibliotheken sind die Komponenten wiederverwendbar an einer zentralen Stelle gespeichert. Dies soll eine konsistente Modellierung unterstützen. Die Artefakte des SysML-Domänenmodells können, durch die Verwendung der «allocate» Beziehung, mit Artefakten des SysML-Systemmodells verknüpft werden.

4.3 Simulationsmodell

Durch die Zusammenführung der Informationen im SysML-Systemmodell wird die Erzeugung von Simulationsmodellen ermöglicht. Ein solches wird verwendet, um die Modellfabrik in einem Tool, welches zur Simulation Digitaler Zwillinge

geeignet ist, zu simulieren. Die Anbindung wird mit Hilfe einer Modelltransformation automatisiert. Dadurch soll der Aufwand und das Fehlerpotential bei der Erstellung eines Simulationsmodells reduziert werden. Ein Profil soll die Transformation erleichtern, indem zu simulierende Komponenten mit dessen Stereotypen versehen werden. Diese stellen durch ihre Tagged Values für das Simulationsmodell relevante Informationen zur Verfügung. Um Verwechslungen mit dem domänenspezifischen Profil aus Abschnitt 4.2 zu vermeiden, wird dieses Profil im Rahmen des Konzeptes Simulationsprofil genannt. Neben den Strukturkomponenten wird in SMs modelliertes Verhalten transformiert. Durch eine M2T Transformation, wird aus den SMs ein simulierbarer Code generiert. Es wird das Simulationsprofil und dessen Stereotypen generisch gestaltet. Teile der Transformation und deren Ausgang, hängen jedoch von der Simulationsumgebung ab. Aus diesem Grund wird die Modelltransformation in diesem Kapitel nur sehr oberflächlich beschrieben. Im Abschnitt 5.3 wird detailliert darauf eingegangen.

4.3.1 Simulationsprofil

Durch das Simulationsprofil soll es möglich sein zu simulierende Komponenten des Systems zu identifizieren. Diese Arbeit wird sich dabei auf Komponenten beschränkt, welche translatorische oder rotatorische Bewegungen ausführen können. Diese Bewegungen sollen in der Simulationsumgebung simuliert werden können. Das Simulationsprofil beinhaltet Stereotypen, welche auf die entsprechenden Blöcke der Systemkomponenten angewendet werden. Tagged Values werden verwendet, um Parameter wie Bewegungsrichtung, Geschwindigkeit oder Beschleunigung anzugeben. Auf das Simulationsprofil wird in Abschnitt 5.3 genauer eingegangen.

4.3.2 Modelltransformation

Das Quellmodell der Modelltransformation ist das SysML- Systemmodell. Die Modelltransformation besteht aus zwei Teilen. Der erste Teil bezieht sich auf die zu simulierenden Strukturelemente. Diese wurden mit Stereotypen des Simulationsprofils versehen und durch Tagged Values mit Informationen ergänzt. Mit Hilfe der Stereotypen werden Elemente des Simulationsmodells erzeugt. In den Tagged Values enthaltene Informationen werde übernommen. Der Zweite Teil bezieht sich auf das Verhalten, welches mit Hilfe von SM modelliert wird. Es wird dabei aus einer SM Code, in Form einer switch-Anweisung, generiert. Dabei werden die SM Elemente *State*, *Transition* und *ChangeEvent* im Code berücksichtigt. Für jene Teile, welche nachträglich ergänzt werden, befinden sich geschützte Bereiche im generierten Code.

5 Modellierung und Simulationsanbindung

In diesem Kapitel wird die Umsetzung des Konzeptes und die benötigten Komponenten beschrieben. Ein großer Teil der Umsetzung wird mit dem Open Source Programmierwerkzeug *Eclipse* und dessen Plugins realisiert. Für die Modellierung in der Modellierungssprache SysML wird das *Papyrus* und für eine M2T Transformation das *Acceleo* Plugin verwendet. Als Simulationsumgebung wird *twin* des Unternehmens Eberle Automatische Systeme GmbH & Co KG [1] verwendet. Eine API und der Open Source Code-Generator *Accele* ermöglichen die Anbindung an twin.

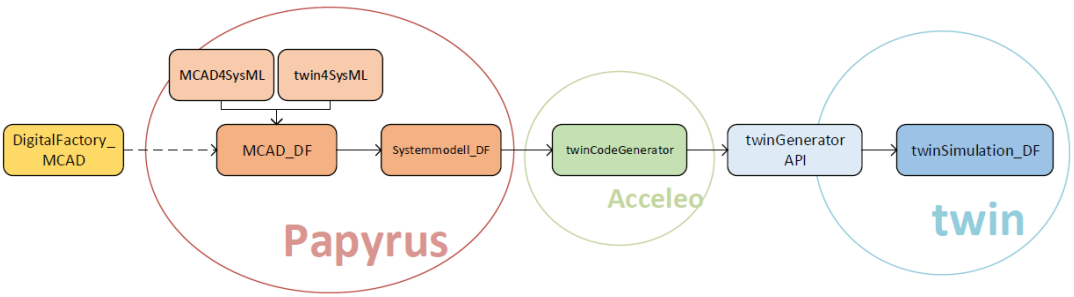


Abbildung 5.1: Umsetzung des Konzepts

In Abbildung 5.1 wird die Verkettung der Komponenten zur Realisierung des Konzepts abgebildet. Das MCAD Modell *DigitalFactory_MCAD* wird mit Hilfe des SysML Profils *MCAD4SysML* in das SysML-Domänenmodell *MCAD_DF* übernommen. Die Artefakte können durch ein weiteres SysML Profil *twin4SysML* mit Informationen für die Simulation ergänzt werden. Die Artefakte werden in das SysML-Systemmodell *Systemmodell_DF* importiert und mit Strukturelementen verknüpft. Dieses Modell ist das Quellmodell einer M2T Transformation, welche im Acceleo Projekt *twinCodeGenerator* implementiert wurde. Der generierte Code wird in der Applikation *twinGenerator* verwendet, um Simulationskomponenten und Verknüpfungen im twin-Projekt *twinSimulation_DF* zu erzeugen.

[1] https://www.eberle.at/de/produkte/twin/

5.1 Systemmodell_DF

Das SysML-Systemmodell wird *Systemmodell_DF* genannt und besteht aus der in Abschnitt 4.1 beschriebenen Top-Level-Struktur. In den folgenden Abschnitten wird auf die wesentlichen Inhalte der darin enthaltenen Packages eingegangen.

5.1.1 Anforderungen

Im Package *Requirements* des Systemmodell_DFs werden Anforderungen an die Modellfabrik modelliert. Für die Arbeit wurden eigens Anforderungen zur Umsetzung und Veranschaulichung des Konzepts konstruiert. Durch folgende Herangehensweise wurden die Anforderungen übersichtlich modelliert:

1. **Gruppierung**
   Anforderungen werden in Gruppen unterteilt. Die Gruppierung wird mit Anforderungsblöcken ohne Text realisiert. Innerhalb der Gruppe wird zusätzlich zwischen den Anforderungsarten *Structural*, *Behavioral* und *Nonfunctional* unterschieden. Innerhalb dieser Aufteilung hat jede Gruppe, Anforderungsart und Anforderung eine eindeutige Nummer.

2. **ID**
   Die ID setzt sich zusammen aus:
   REQ-‹GruppenNR›.‹AnforderungsartNR›.‹AnforderungsNR›

3. **Abstraktion**
   Kann eine Anforderung auf einer niedrigeren Abstraktionsebene beschrieben werden, wird dies in einer weiteren Gruppe realisiert. Diese wird über die «DeriveReqt» Beziehung mit der Anforderung höheren Abstraktionsniveaus verknüpft.

In Abbildung 5.2 wird die Herangehensweise in einem Anforderungsdiagramm dargestellt. In der Gruppe *REQ-0.0.0 Digital Factory* gibt es drei Anforderungen der Art *Structural Requirements*. Diese können in weiteren Gruppen detaillierter beschrieben werden.

Wie bereits in Unterabschnitt 4.1.1 erläutert, gehören Anforderungen und Entwurf eines Systems zusammen. In Abbildung 5.3 wird dargestellt, dass die Anforderung *REQ-0.0.0* von einem Part mit dem Namen *fertigungsanlage* erfüllt werden soll. Da ein Part die Instanz eines Blocks in SysML darstellt, ist auch dieser in der Regel für die Erfüllung einer Anforderung verantwortlich, und

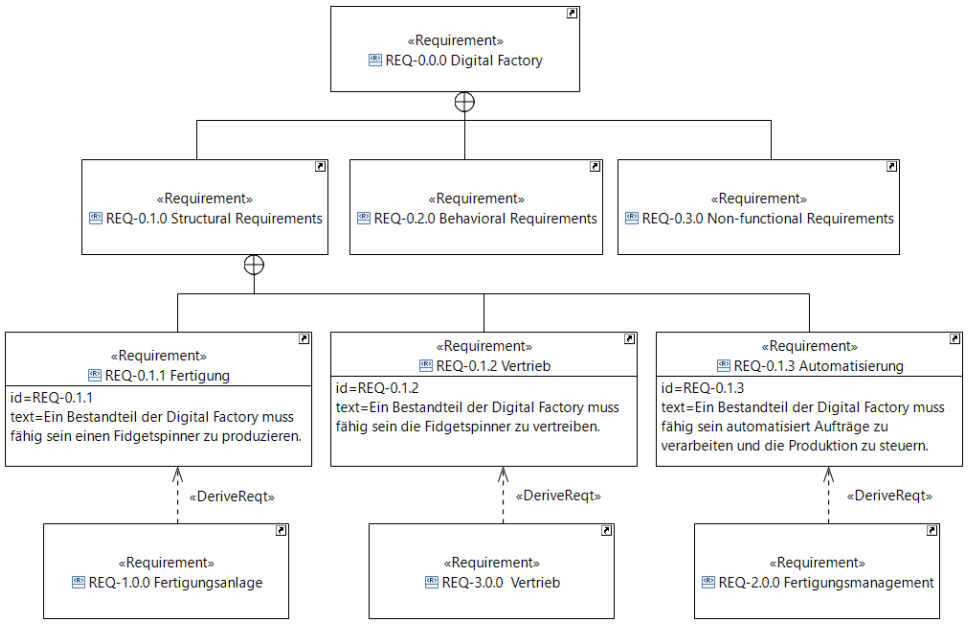


Abbildung 5.2: Anforderungsdiagramm

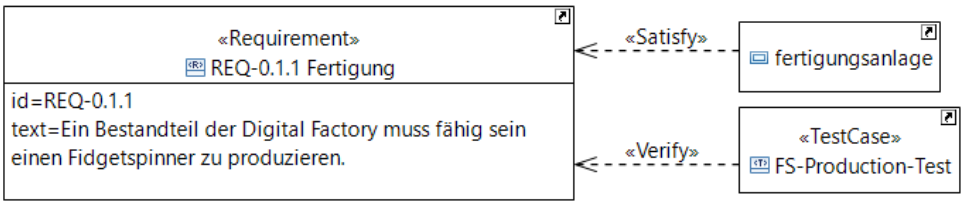


Abbildung 5.3: Verwendung der «Satisfy» und «Verify» Beziehung

nicht der Block selbst. Der Test *FS-Production-Test* soll die tatsächliche Erfüllung der Anforderung überprüfen. Zur Sammlung aller «Satisfy» und «Verify» Beziehungen, wird im Package *DigitalFactory* das Package *Allocations* mit Subpackage *requirement allocation* erstellt.

Zur visuellen Kontrolle und Übersicht können Anforderungen und ihre Beziehungen zu anderen Artefakten in Tabellen dargestellt werden. In Abbildung 5.4 ist eine Anforderungstabelle abgebildet. Diese kann in Papyrus automatisch erstellt werden und beinhaltet alle Anforderungen, welche sich im selben Package wie die Tabelle befinden. Durch die beschriebene Strukturierung der Anforderungen, können alle Ebenen in einer Tabelle dargestellt werden. Es wurden die Spalten *satisfiedBy* und *verifiedBy* hinzugefügt, um die *satisfy* und *verify* Beziehungen anzuzeigen.

	id	text	/satisfiedBy	/verifiedBy
REQ-0.0.0 Digital Factory	REQ-0.0.0			
REQ-0.1.0 Structural Requirements	REQ-0.1.0			
REQ-0.1.1 Fertigung	REQ-0.1.1	Ein Bestandteil der Digital Factory muss fähig sein einen Fidgetspinner zu produzieren.	fertigungsanlage	FS-Production-Test
REQ-0.1.2 Vertrieb	REQ-0.1.2	Ein Bestandteil der Digital Factory muss fähig sein die Fidgetspinner zu vertreiben.	vertrieb	
REQ-0.1.3 Automatisierung	REQ-0.1.3	Ein Bestandteil der Digital Factory muss fähig sein automatisiert Aufträge zu verarbeiten und die Prod..	fertigungsmanagement	
REQ-0.2.0 Behavioral Requirements	REQ-0.2.0			
REQ-0.3.0 Non-functional Requirements	REQ-0.3.0			
REQ-1.0.0 Fertigungsanlage	REQ-1.0.0			
REQ-2.0.0 Fertigungsmanagement	REQ-2.0.0			
REQ-3.0.0 Vertrieb	REQ-3.0.0			
REQ-4.0.0 Bauteilbearbeitung	REQ-4.0.0			

Abbildung 5.4: Anforderungstabelle

5.1.2 Struktur

Die Struktur des Systems wird im Package *DigitalFactory::Structure* mit den in Unterunterabschnitt 4.1.2.1 beschriebenen Konzepten modelliert. Sie wird durch Packages in drei Ebenen unterteilt, um die Modellierung übersichtlicher zu gestalten. Die oberste Ebene *System Level* definiert das System und die Beziehung zu dessen Subsystemen. Die mittlere Ebene *Subsystem Level* definiert alle Subsysteme und deren Beziehungen. Die unterste Ebene *Components Level* definiert alle Komponenten, welche in domänenespezifischen Tools modelliert und anschließend realisiert werden sollen. In Abbildung 5.5 wird die Struktur

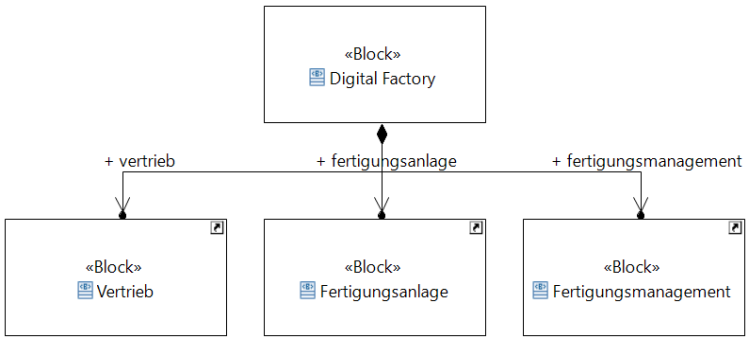


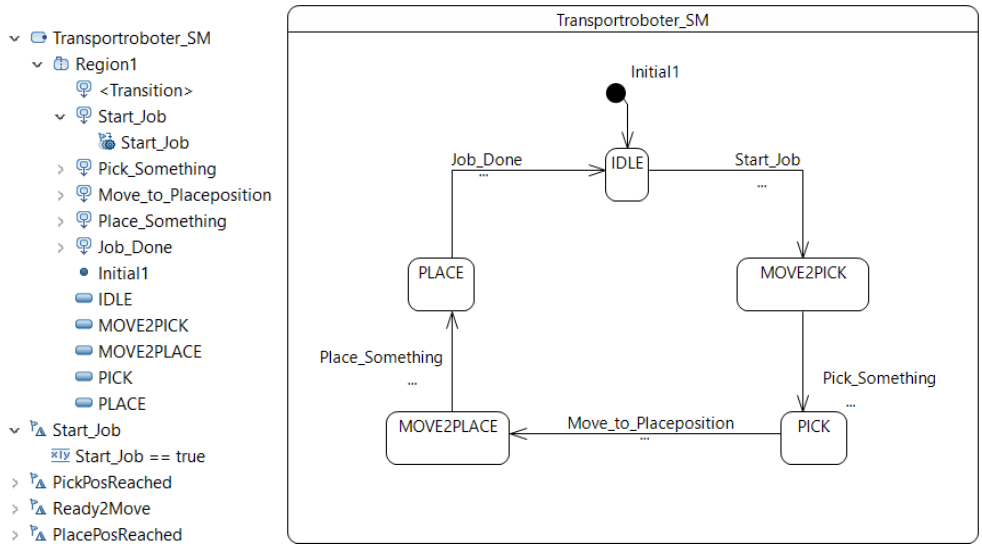
Abbildung 5.5: System Level

auf oberster Ebene abgebildet, um die angewendeten Konzepte zu veranschaulichen. Es wird der Block *Digital Factory* definiert, welcher drei Parts mit den Namen *vertrieb*, *fertigungsanlage* und *fertigungsmanagement*, sowie den jeweiligen Typen besitzt. Die Blöcke *Vertrieb*, *Fertigungsanlage* und *Fertigungsmanagement* sind Subsysteme und können in weitere Blöcke unterteilt werden.
Die Auswirkung, welche die beschriebene Modellierung auf den Block Digital Factory hat, wird in Abbildung 5.6 gezeigt. Sie zeigt den Block und die Verschachtelung seiner Part Properties im Model Explorer.

Abbildung 5.6: Digital Factory Block

5.1.3 Verhalten

Manche Komponenten der Digital Factory sind aktiv und weisen ein Verhalten auf, welches simuliert werden kann. Dieses kann im SysML-Systemmodell im Package *DigitalFactory::Behavior* mit SMs modelliert werden. In Abbildung 5.7 wird für den Transportroboter eine vereinfachte *SM* abgebildet. Die SM enthält insgesamt fünf *Zustände (states)*, welche über *Transitionen (transitions)* miteinander verbunden werden. Jede Transition besitzt einen *Trigger*, welcher sie einleitet. Diese werden als *Change Events* modelliert. In diesen Change Events kann ein Ausdruck, beispielsweise in Form einer *Opaque Expression*, definiert werden, welcher zum Auslösen des Events führt. 5.7a zeigt die SM *Transportroboter_SM* im Model Explorer mit ihren Elementen. In 5.7b wird die SM in einem State Machine Diagram dargestellt. Dieses Verhalten wird mit einer «allocate» Beziehung mit dem Block Transportroboter verknüpft. Zur Sammlung der Zuweisungen zwischen Struktur und Verhalten wird im Package *DigitalFactory::Allocations* ein Subpackage *behavioral allocation* erstellt.



(a) Model Explorer (b) State Machine Diagram

Abbildung 5.7: Transportroboter State Machine

5.2 MCAD_DF

Das SysML-Domänenmodell wird *MCAD_ DF* genannt und beinhaltet ein Package mit dem Namen *MCAD Components Library* und dem Stereotype «ModelLibrary». In dieses Package wird das MCAD Modell der Digital Factory, welches in Solidworks erstellt wurde, übernommen. Ein solches MCAD Modell besteht aus einem Assembly, welches wiederum aus Subassemblies und Parts besteht. Für die domänenspezifische Modellierung mit SysML wurde das Profil *MCAD4SysML* erstellt, welches in Abbildung 5.8 abgebildet ist.

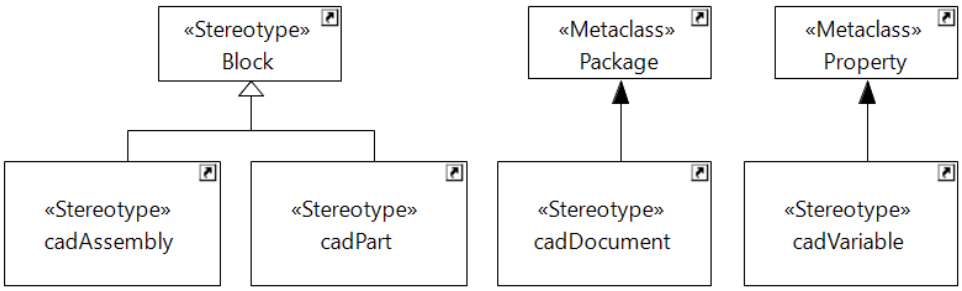


Abbildung 5.8: MCAD4SysML

Es besitzt die Stereotypen *cadAssembly* und *cadPart*, welche Generalisierungen des Stereotyps Block sind. Mit dem Stereotyp *cadVariable* können dazugehörige Variablen aus dem MCAD Dokument exportiert werden. Durch den

Stereotyp *cadDocument* können die Artefakte gruppiert und mit Name und Pfad des MCAD Dokuments gespeichert werden. In Abbildung 5.9 wird das MCAD_DF Modell im Model Explorer abgebildet. Für das Assembly auf der höchsten Ebene im MCAD Modell wird ein Package mit dem Stereotyp «cadDocument» erstellt. Es enthält die Baugruppe selbst, sowie alle Subassemblies und Parts. Für jedes Subassembly wird zur Übersicht ein eigenes Package erstellt.

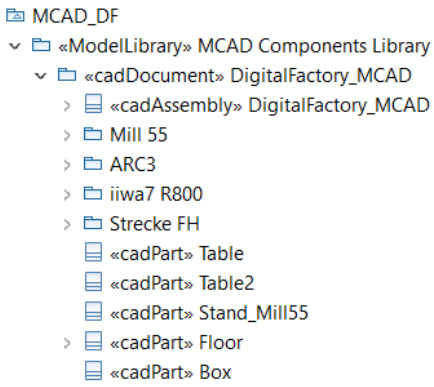


Abbildung 5.9: MCAD_DF Model Explorer

Das Profil wird in das MCAD_DF Modell importiert und kann dadurch zur Modellierung verwendet werden. Mit den Stereotypen des MCAD4SysML Profils wurde das MCAD Dokument aus Abbildung 5.10 in das *MCAD_DF* Modell übertragen. Der Name der Komponenten in einem Solidworks Assembly setzt sich dabei aus *Komponenten Name* und *Instanz ID* zusammen.

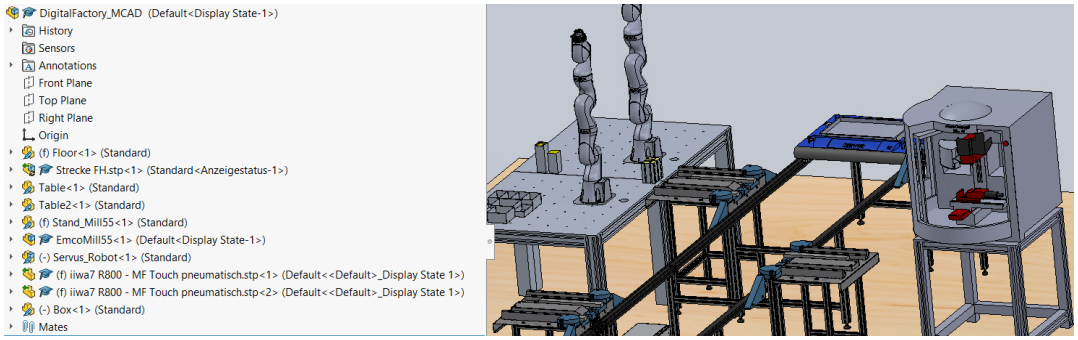


Abbildung 5.10: MCAD Modell in Solidworks

Die höchste Hierarchieebene dieses Assemblies wird in Abbildung 5.11 mit Hilfe eines Block Definition Diagrams dargestellt. Für die Komponenten des Assemblies wurden Blocks mit dem Namen der Komponente und dem entsprechenden Stereotype des Profiles *MCAD4SysML* definiert. Außerdem wurde die Hierarchie des MCAD Modells mit Hilfe der Containment Association übernommen.

Der Name der Part Properies ist wie in Solidworks Komponenten Name und Instanz ID. Das Package *MCAD Components Library* wird anschließend in das *Systemmodell_DF* importiert und kann dort verwendet werden.

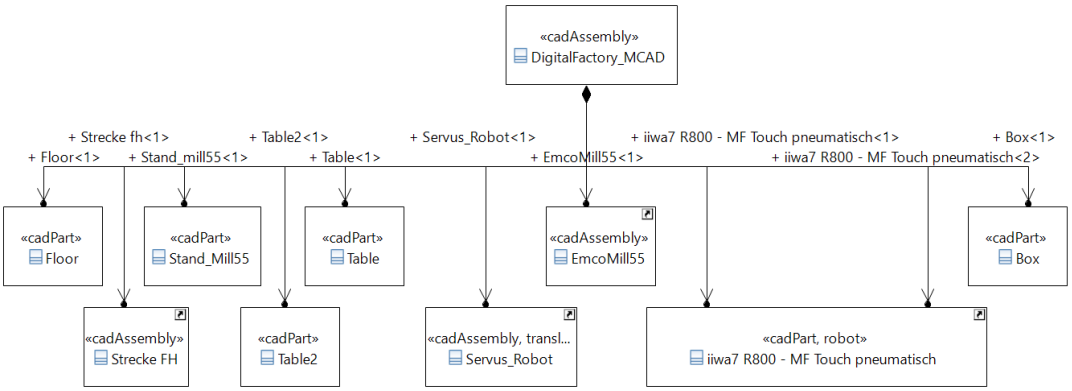
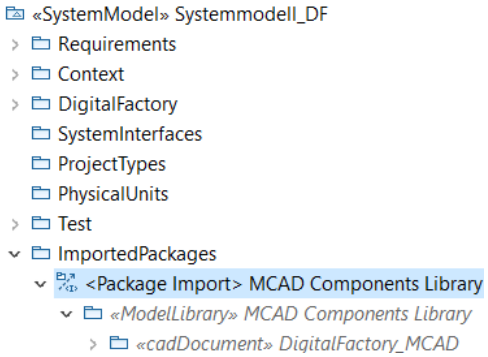


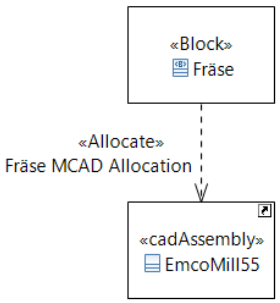
Abbildung 5.11: MCAD Modell in SysML

5.2.1 Import und Anwendung der Bibliothek

In Abbildung 5.12a wird gezeigt, wie die Bibliothek aus dem *MCAD_DF* Modell in das *Systemmodell_DF* importiert wird. Die Artefakte der Bibliothek können nun im Modell verwendet werden. Durch die *Allocate*-Beziehung aus Abbildung 5.12b wird gezeigt, dass der Block *Fräse* aus dem Modell Systemmodell_DF, dem cadAssembly *EmcoMill55* im Modell MCAD_DF entspricht. Zur Sammlung der Zuweisungen dieser Art wird im Package *DigitalFactory::Allocations* ein Subpackage *logical-physical allocation* erstellt.



(a) Import



(b) Anwendung

Abbildung 5.12: Import und Anwendung der Bibliothek

5.3 twinSimulation_DF

Als Simulationssoftware wird *twin* von Eberle Automatische Systeme GmbH & Co KG verwendet. Zur Visualisierung in twin wird ein MCAD Modell im *STandard for the Exchange of Product model data (STEP)* Format importiert. Die darin enthaltene Struktur muss für die Simulation geändert werden, damit kinematische Ketten entstehen. Anschließend werden Simulationskomponenten und deren Verknüpfungen verwendet, um die Logik des Simulationsmodells zu definieren. Eine Simulationskomponente besitzt Inputs, Outputs und Parameter. Outputs können mit Inputs von anderen Komponenten verknüpft werden.

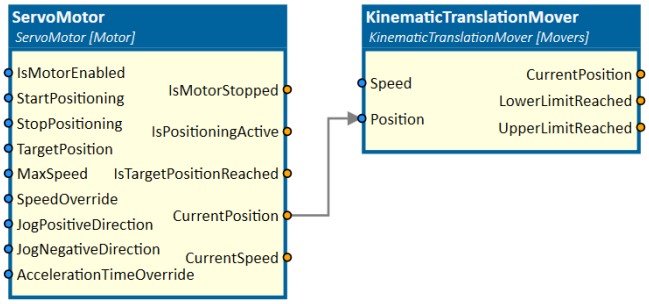


Abbildung 5.13: Simulationskomponenten und Verknüpfungen

In Abbildung 5.13 wird die Simulationskomponente *KinematicTranslationMover* abgebildet. Damit können Objekte translatorisch bewegt werden, indem eine Position oder eine Geschwindigkeit über den jeweiligen Input gesetzt wird. In dieser Abbildung wird die Position durch den Output *CurrentPosition* der Simulationskomponente *ServoMotor* vorgegeben. Während der Simulation läuft im Hintergrund ein Physics Engine und ermöglicht so eine realitätsnahe Simulation von physikalischen Prozessen.

In Abbildung 5.14 wird ein Simulationsmodell in twin dargestellt. Auf der linken Seite wird die Struktur der Baugruppe abgebildet. Im *3D-View* Fenster wird die Baugruppe visualisiert und simuliert. Im *Simulation Components Diagram* können Simulationskomponenten aus einer Bibliothek erzeugt und Verknüpfungen hergestellt werden. Je komplexer die Anlage, desto größer ist der Aufwand, sowie das Fehlerpotential bei der Erstellung und Verknüpfung dieser Komponenten. Durch das Profil *twin4SysML* werden zu simulierende Artefakte aus dem *MCAD_DF* Modell mit Stereotypen versehen. Diese werden in einer M2T-Transformation verwendet, um in Kombination mit einer API Simulationskomponenten und deren Verknüpfungen in twin zu erzeugen.

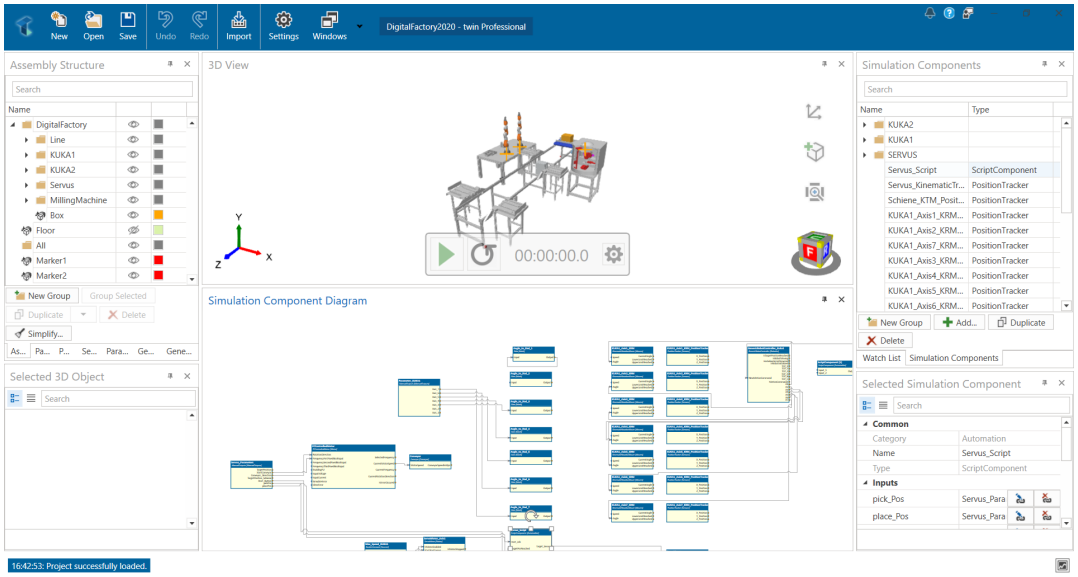


Abbildung 5.14: twin Simulationsumgebung

5.3.1 twin4SysML-Profile

In der Digital Factory gibt es Komponenten, welche auf verschiedene Art beweglich sind. Ein Beispiele ist der Servus Roboter, welcher sich auf dem Schienennetz linear bewegt. Ein Anderes Beispiel wäre die Türe der Fräse, welche um eine Achse rotiert. Ein Letztes Beispiel wären die Kuka Roboter, welche mit Hilfe ihrer Rotationsachsen in der Lage sind, verschiedene PTP und Linearbewegungen durchzuführen. Für diese drei Anwendungsfälle wurden die in Abbildung 5.15 abgebildeten Stereotypen in einem Profil namens *twin4SysML* erstellt. Der Stereotyp *translationalPart* ist für linear bewegliche Teile, *rotationalPart* für rotierende Teile und *robot* für Roboter wie der Kuka Roboter. Sie beinhalten einige Tagged Values, wie beispielsweise *LocalTrannslationAxis* beim Stereotyp translationalPart, welche bei der Anwendung des Stereotyps gesetzt werden können. Das Profil wird in das Modell *MCAD_DF* importiert und dort bei den entsprechenden Blöcken angewendet.

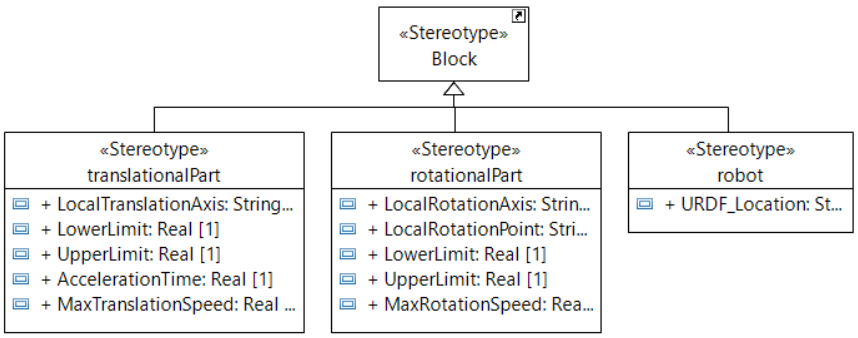


Abbildung 5.15: twin4SysML Stereotypen Definition

5.3.2 twinGenerator

Eberle Automatische Systeme GmbH & Co KG stellt eine Schnittstelle zur Verfügung, um die Generierung des Simulationsmodells zu erleichtern. Diese API ermöglicht die Erzeugung von Simulationskomponenten und deren Verknüpfungen in der Programmiersprache C#. Eine Komponente kann mit Code im twinGenerator wie in Abbildung 5.16 erzeugt werden. In diesem Beispiel wird die Simulationskomponente *KinematicTranslationMover* erzeugt und die Parameter *Name*, *LowerLimit* und *UpperLimit* gesetzt.

```
var c3 = testEnvironment.CreateComponent<V3S.ComponentLibrary.Basic.Movers.KinematicTranslationMover>();
componentContainer.Add(c3);
c3.Name = "Servus_KTM";
c3.LowerLimit = 0;
c3.UpperLimit = 5000;
```

Abbildung 5.16: Erzeugung einer Komponente mit dem TwinGenerator

Die Verknüpfungen werden mit den Zeilen aus Abbildung 5.17 erstellt. Es wird der Eingang *Position* der Variable *c3* mit dem Ausgang *CurrentPosition* der Variable *c4* verbunden und in der Liste *linkedInputs* gespeichert. Mit Hilfe einer M2T Transformation wird C# Code wie dieser für die zu simulierenden Artefakte erzeugt werden.

```
var inPos = c3.Inputs.Single(i => i.Name == nameof(c3.Position));
var outPos = c4.Outputs.Single(o => o.Name == nameof(c4.CurrentPosition));
outPos.LinkInput(inPos);
linkedInputs.Add(IOLinkInfo.CreateFrom(inPos));
```

Abbildung 5.17: Erzeugung einer Verknüpfung mit dem TwinGenerator

5.3.3 M2T Transformation mit Acceleo

Die M2T Transformation wird mittels Acceleo in Eclipse umgesetzt. In Abbildung 5.18 ist der Projektordner des Acceleo Projects *twinCodeGenerator* abgebildet. Der Ordner *src* beinhaltet dabei die für die Transformation benötigten *Acceleo Module Files*, welche auf *.mtl* enden. Diese können ausgeführt werden und generieren aus einem definierten Quellmodell Files in einem Zielordner. Quellmodell ist das SysML-Modell *Systemmodell_DF* und Zielordner der Ordner *gen_Files* des Acceleo Projects.

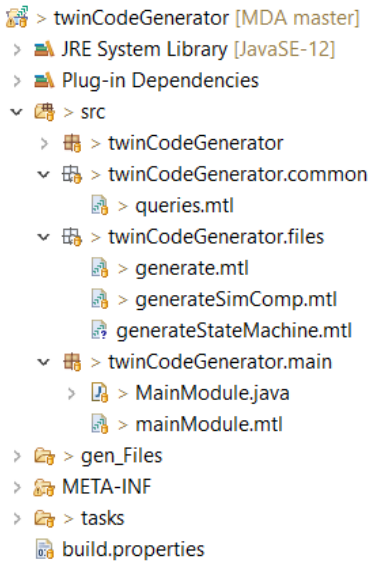


Abbildung 5.18: twinCodeGenerator Acceleo Project

5.3.3.1 Transformationstruktur

Die Transformation wird auf die in Abbildung 5.18 abgebildeten Module aufgeteilt. Es gibt ein Modul mit dem Namen *mainModule*, welches ein Template mit dem Namen *mainTemplate* besitzt. Dies ist der Einstiegspunkt der Transformation und beinhaltet die Navigation durch das Quellmodell. Ein Module mit dem Namen *generate* enthält Templates, welche ein *.cs-File* für C# Code des twinGenerators und ein *.txt-File*, für Zusatzinformationen, initialisieren. Das sind Informationen, welche aktuell vom twinGenerator und er API nicht verarbeitet werden können und die Nachbearbeitung in twin erleichtern sollen. Der Inhalt dieser Files wird mit den Templates aus dem Modul *generateSimComp* generiert. Das Modul *generateStateMachine* ist für die Generierung eines *.cs-Files* mit Code zur Simulation einer SM zuständig. In dem Modul *queries* sind Queries zur Unterstützung der Transformation enthalten.

```
1  [comment encoding = UTF-8 /]
2  [module mainModule('http://www.eclipse.org/uml2/5.0.0/UML',
3
4  [import twinCodeGenerator::files::generate/]
5  [import twinCodeGenerator::common::queries/]
6  [import twinCodeGenerator::files::generateStateMachine /]
7
8  [template public mainTemplate(aModel : Model)
9  {
10     fileName:String = 'TwinComponents';
11 }
12 ]
13 [comment @main/]
```

Abbildung 5.19: mainModule.mtl

In Abbildung 5.19 wird ein Teil des Modules *mainModule* abgebildet. Bei dessen Definition werden die URIs zu den verwendeten Metamodellen als Parameter angegeben (Zeile 2). Das Module besizt ein Template *mainTemplate*, welches auf Artefakte vom Typ *Model* angewendet werden kann (Zeile 8). In Zeile 10 wird die Variable *fileName* deklariert und mit dem String *TwinComponents* initialisiert. Diese Variable kann im gesamten Template verwendet, der Wert jedoch nicht geändert, werden. Mit *[comment @main/]* aus Zeile 13 wird signalisiert, dass es sich um ein *main template* handelt. Diese bilden den Einstiegspunkt für eine Transformation. Das Template mainTemplate wird für alle Artefakte des Quellmodells mit dem Typ *Model* ausgeführt. In den Zeilen 4 bis 6 werden zusätzlich benötigte Module importiert.

5.3.3.2 Transformationsablauf

Zur Durchführung einer Transformation wird das File *mainModule.mtl* ausgeführt. In diesem wird durch das Quellmodell zu den Artefakten mit den Stereotypen des *twin4SysML* Profiles und zu den modellierten SMs navigiert. Der Transformationsablauf ist dabei folgender:

1. Initialisiere Files *TwinComponents.cs* und *TwinComponents.txt*
2. Wähle die Allocations im Package "logical-physical allocation"
3. Für alle Supplier: Generiere Simulationskomponenten für jene Elemente des MCAD Dokuments vom Typ des Suppliers
4. Wähle die Allocations im Package "behavioral allocation"
5. Generiere Script Komponente inklusive ScriptText für jede SM

Schritt 1 wird mit dem Template *generateTwinComponentsFiles* aus dem Module *generate* durchgeführt. Mit dem *file*-Block kann in Acceleo ein File erstellt und/oder beschrieben werden. Mit dem Paramter *false* wird ein neues File erstellt und mit *true* ein bestehendes ergänzt. Das Template ist in Abbildung 5.20 abgebildet.

```
[template public generateTwinComponentFiles(aModel:Model, fileName: String)]
[file (fileName + '.cs', false, 'UTF-8')]
    /* This File includes the twin Simulation Components of the Model "[aModel.name/]" */
[/file]
[file (fileName+'.txt', false, 'UTF-8')]
********************************************************************************
    This File includes additional information to File "[fileName + '.cs'/]"
********************************************************************************
[/file]
[/template]
```

Abbildung 5.20: Transformation Schritt 1

Schritt 2 der Transformation wird in Abbildung 5.21 abgebildet. Mit dem ersten *for*-Block wird über alle enthaltenen Artefakte vom Typ *Package* im Modell *aModel*, welche den Namen *logical-physical allocation* besitzen, iteriert. Das ist jene Stelle im Modell, an welcher die Allocations aus Unterabschnitt 5.2.1 zur Einbindung der Bibliothekskomponenten gespeichert werden. Mit dem zweiten for-Block wird anschließend in diesem Package durch alle Artefakte vom Typ *Abstraction* iteriert. Da *Allocate* ein SysML Stereotyp ist, welcher den UML Typ *Abstraction* erweitert, werden alle Allocations des Packages ausgewählt.

```
[comment 2) Wähle Allocations im Package mit dem Namen "logical-physical allocation"/]
[for(aPackage : Package| aModel.eAllContents(Package)->select(p|p.name = 'logical-physical allocation'))]
    [for(aAbstraction : Abstraction | aPackage.eContents(Abstraction))]
```

Abbildung 5.21: Transformation Schritt 2

Schritt 3 ist für die Generierung der Simulationskomponenten zuständig. Die Supplier der Allocations aus Schritt 2 sind die Elemente der importierten Library. Es wird nun für jeden Supplier im dazugehörigen MCAD Dokument nach Parts vom selben Typ gesucht. Für diese wird das Template *generateElements* des Moduls *generate* aufgerufen. Die Navigation im Module *mainModule* wird in Abbildung 5.22 abgebildet. Das Query *getMainAssembly*, welches in der dritten Zeile verwendet wird, ermittelt das Top Level Assembly im MCAD Dokument des jeweiligen Suppliers. Im Template generateElements wird für die jeweilige Klasse und alle seine Properties vom Typ Klasse das Template *generateTwinComponents* im Module *generateSimComp* aufgerufen. Dadurch werden auch die Part Properties der Bibliothekskomponenten, wie beispielsweise die Türe der Fräse, berücksichtigt.

```
[comment 3) Generiere Simulationskomponenten für alle Parts des Dokuments mit den Supplier Typen/]
        [for(aElement: NamedElement | aAbstraction.supplier->filter(uml::Class))]
            [for(aClass:Class | aElement.getMainAssembly())]
                [for(aProperty:Property | self.eAllContents(Property)->select(a|a.type = aElement))]
                    [aElement.oclAsType(Class).generateElements(fileName,aProperty.name)/]
```

Abbildung 5.22: Transformation Schritt 3

In Abbildung 5.23 wird der Inhalt des Modules *generateSimComp* abgebildet. Mit Hilfe von *Guards* wird je nach Stereotyp das richtige Template ausgewählt. Das Query *hasStereotype* überprüft dabei, ob die Klasse den entsprechenden Stereotype besitzt. In den Templates des Modules wird je nach Stereotype Inhalt in den Files *TwinComponents.cs* und *TwinComponents.txt* generiert.

```
[comment encoding = UTF-8 /]
[module generateSimComp('http://www.eclipse.org/uml2/5.0.0/UML', 'http://www.eclipse.org/papyrus/sysml/1.6/SysML', 'http://www.

[import twinCodeGenerator::common::queries/]

[template public generateTwinComponents(aClass : Class,fileName:String, partName: String) ? (hasStereotype('translationalPart')
[/template]

[template public generateTwinComponents(aClass : Class,fileName:String, partName: String) ? (hasStereotype('rotationalPart'))
[/template]

[template public generateTwinComponents(aClass : Class,fileName:String, partName: String) ? (hasStereotype('robot'))
[/template]
```

Abbildung 5.23: Module generateSimComp

In Abbildung 5.24 wird ein Teil des Templates abgebildet, welches Code für Komponenten mit dem Stereotyp *translationalPart* generiert. Für jede zu erstellende Simulationskomponente wird eine String Variable definiert, welche den Namen der Komponentenvariable im File *twinComponents.cs* ermittelt. Die Definition der Variablen für die Simulationskomponenten des Stereotyps translationalPart wird in den Zeilen 8-14 abgebildet. Der Name von Komponentenvariablen wird dynamisch erzeugt und setzt sich aus dem Kleinbuchstaben *c*, einer eindeutigen *ID* und einer komponentenspezifischen Endung zusammen. Die eindeutige ID wird mit dem Query *getElementID* für die aufrufende Klasse ermittelt. Somit wird verhindert, dass Variablen mehrfach im File twinComponents.cs definiert werden. In den Zeilen 25-27 wird mit dem Query *getTaggedValue* auf die Tagged Values des Stereotyps *translationalPart* zugegriffen.

Das Template *generateTwinComponents* generiert Code im File *twinComponents.cs*. Der entsprechende Output wird in Abbildung 5.25 dargestellt. Es wird die Simulationskomponente *KinematicTranslationMover* erstellt, und dazugehörige Parameter, wie *LocalTranslationAxis*, *LowerLimit* und *UpperLimit*, gesetzt.

```
[template public generateTwinComponents(aClass : Class,fileName:String, partName: String) ? (hasStereotype('translationalPart'))
{
    ktm:String = 'c'+ aClass.getElementID().toString() + '_KTM';
    servo:String = 'c'+ aClass.getElementID().toString() + '_Servo';
    tru:String = 'c' + aClass.getElementID().toString()+'_TRUE';
    maxV:String ='c' + aClass.getElementID().toString()+'_MAXV';
    mOut:String = 'c' + aClass.getElementID().toString()+'_MOUT';
    inVar:String = 'in_c'+aClass.getElementID().toString();
    outVar:String = 'out_c'+aClass.getElementID().toString();
}]

[file (fileName + '.cs' ,true, 'UTF-8')]
/*
Simulation Components declaration for translationalPart [partName/]
*/

    var [ktm/] = testEnvironment.CreateComponent<V3S.ComponentLibrary.Basic.Movers.KinematicTranslationMover>();
    componentContainer.Add([ktm/]);
    [ktm/].Name = "[partName.toUpperFirst().replaceAll(' ', '') + '_KTM'/]";
    [ktm/].LocalTranslationAxis = new Vector3[self.getTaggedValue('twin4SysML::translationalPart','LocalTranslationAxis')/];
    [ktm/].LowerLimit = [self.getTaggedValue('twin4SysML::translationalPart','LowerLimit')/];
    [ktm/].UpperLimit = [self.getTaggedValue('twin4SysML::translationalPart' ,'UpperLimit')/];
    [ktm/].ControllingType=V3S.ComponentLibrary.Basic.Movers.TypeOfTranslationControlling.Position;
```

Abbildung 5.24: Acceleo Template für Stereotyp *translationalPart*

```
/*
Simulation Components declaration for translationalPart Servus_Robot<1>
*/

var c72_KTM = testEnvironment.CreateComponent<V3S.ComponentLibrary.Basic.Movers.KinematicTranslationMover>();
componentContainer.Add(c72_KTM);
c72_KTM.Name = "Servus_Robot<1>_KTM";
c72_KTM.LocalTranslationAxis = new Vector3(1, 0, 0);
c72_KTM.LowerLimit = 0.0;
c72_KTM.UpperLimit = 5000.0;
c72_KTM.ControllingType = V3S.ComponentLibrary.Basic.Movers.TypeOfTranslationControlling.Position;
```

Abbildung 5.25: KinematicTranslationMover im File *twinComponents.cs*

Schritt 4 ist/erzeugt eine Wiederholung von Schritt 2. Anstelle des Packages *logical-physical allocation* wird jedoch das Package *behavioral allocation* gewählt.

Schritt 5 ist für die Codegenerierung der modellierten SMs zuständig. Dies wird mit den Templates des Modules *generateStateMachine* realisiert, welche in Abbildung 5.26 abgebildet sind. Mit dem Template *generateTwinScriptComp* wird die Simulationskomponente *ScriptComponent* im File *TwinComponents.cs* erstellt. Diese kann verwendet werden, um der Simulation benutzerdefiniertes Verhalten hinzuzufügen. Im Parameter *ScriptText* der Simulationskomponente kann C# Code gespeichert werden, welcher periodisch ausgeführt wird. Das Template *generateScriptText* erzeugt Teile dieses Codes und speichert ihn in ein File, welches mit dem Template *initScript* erzeugt wird. Inputs, Outputs und statische Variablen des Scripts können in der Simulationskomponente deklariert und initialisiert werden. Es werden für jede SM die Variablen *time* vom Typ *Double* und *state* vom Typ *Interger* deklariert und mit den Werten *0.0* und *0* initialisiert. Die Variable state wird den aktuellen State speichern und die Variable time die Zeit, welche sich die SM bereits im aktuellen State befindet.

Mit Ausnahme der Variablen *state* und *time* werden keine Variablen, Inputs oder Outputs durch die Transformation erzeugt. Diese müssen nachträglich in der Simulationssoftware hinzugefügt werden, damit die Simulationskomponente *ScriptComponent* mit anderen Simulationskomponenten im *Simulation Components Diagram* verknüpft werden kann.

```
[comment encoding = UTF-8 /]
[module generateStateMachine('http://www.eclipse.org/uml2/5.0.0/UML', 'http://www.eclips

[import twinCodeGenerator::common::queries/]

[template public generateTwinScriptComp(aStateMachine : StateMachine, fileName : String)
[/template]

[template public initScript(aStateMachine : StateMachine, fileName : String)]
[/template]

[template public generateScriptText(aStateMachine : StateMachine, fileName: String)]
[/template]

[template public generateTransitions(aState: State, fileName: String)
```

Abbildung 5.26: Module generateStateMachine

Zusätzlich wird die Verknüpfung dieser Inputs mit lokalen Variablen, sowie das Setzen der Outputs, nachträglich im SkriptText ergänzt. Dafür werden geschützte Bereiche im generierten Code bereitgestellt. Diese Bereiche bleiben erhalten und werden bei einer erneuten Codegenerierung nicht überschrieben. In Abbildung 5.27 wird die Generierung eines solchen Bereichs gezeigt.

```
//[protected ('Inputs')]
    //TODO: Link Inputs to Change Event Variables
    //z.B. var start = Inputs.Get<bool>(0);
//[/protected]
```

Abbildung 5.27: geschützte Bereiche im Code

Aus allen States der SM wird zu Beginn des Files eine Enumeration erstellt. Diese wird verwendet um anschließend die Navigation im Code zu erleichtern. Der dazugehörige Transformations-Code wird in Abbildung 5.28 abgebildet.

```
//States
enum States
{[/file]
[for(aRegion: Region | self.eContents(Region))]
    [for(aState: State | self.eContents(State))]
        [if(self = aRegion.eContents(State)->last())]
            [file (fileName + '.cs', true, 'UTF-8')][self.name/][/file]
        [else]
            [file (fileName + '.cs', true, 'UTF-8')][self.name/],[/file]
        [/if]
```

Abbildung 5.28: Erstellen einer Enumeration für States

Für alle in der SM verwendeten Change Events wird eine lokale Variable erstellt, deren Wert unter anderem durch die im Change Event enthaltene Opaque Expression bestimmt wird. Dies wird in Abbildung 5.29 dargestellt. Mit der Bedingung *(time>SimulationWorld.SimulationTimeStep)* wird verhindert, dass unmittelbar nach einer Transition eine weitere durchgeführt wird. Damit wird erreicht, dass sich im State gesetzte Outputs auf das Change Event auswirken, bevor dieses eine Transition auslösen kann.

```
[for(aChangeEvent: ChangeEvent | self.eContainer(Package).eAllContents(ChangeEvent))]
[file (fileName + '.cs', true, 'UTF-8')]
var ev_[self.name/] = (([self.eContents(OpaqueExpression)->first()._body/])&&(time>SimulationWorld.SimulationTimeStep))? true:false;
```

Abbildung 5.29: Erstellen der Change Event Variablen

Anschließend wird eine *switch*-Anweisung eingefügt. Für jeden State wird ein *case* mit einem geschützten Bereich für Benutzerdefinierten Code erzeugt. Dies wird in Abbildung 5.30 abgebildet.

```
switch(state)
{[/file]
    [for(s: State | self.eContents(State))]
[file (fileName + '.txt', true, 'UTF-8')]
    case (int)States.[self.name/]:
            //[protected (self.name)]
                //TODO: Code for State [self.name/] should be implemented
            //[/protected]
            [generateTransitions(self, fileName)/]
            break;
```

Abbildung 5.30: Skript switch-Anweisung

Außerdem werden mit dem Template *generateTransitions* Transitions erstellt. Es handelt sich dabei um *if*-Anweisung, welche für alle Transitions erstellt werden, bei welchen der aktuelle State die *Source* ist. Die Anweisung überprüft die entsprechende Variable des zuständigen Change Events und wechselt den State zum *Target*-State der Transition. Dies wird in Abbildung 5.31 gezeigt.

```
[for(aTransition: Transition | sm.eAllContents(Transition)->select(t | t.source.name = self.name))]
    [for(aTrigger : Trigger | aTransition.eContents(Trigger))]
        [for(aEvent: Event | sm.eContainer(Package).eAllContents(Event)->select(i | i.name = self.event.name))]
            if (ev_[self.name/])
            {
                state = (int)States.[aTransition.target.name/];
            }
```

Abbildung 5.31: Skript Transitionen

6 Ergebnis am Beispiel der Digital Factory

Der Ablauf der Arbeit wird in diesem Kapitel anhand einer Komponente aus der Digital Factory beispielhaft veranschaulicht. Abbildung 6.1 stellt den gesamten Ablauf des Ansatzes dar. Die Modellierung beginnt im SysML Modell *Systemmodell_DF* mit der Modellierung von Spezifikation, Entwurf und Verifikation. Die physikalische Struktur des Entwurfs wird im MCAD Modell *DigitalFactory_MCAD* konkretisiert. Der Inhalt dieses Modells wird in das Modell *MCAD_DF* übernommen und in einer Model Library *MCAD Components Library* gespeichert. Diese wird im Systemmodell_DF importiert und mit Strukturkomponenten verknüpft. Das Verhalten dieser Komponenten wird als SM modelliert und ebenfalls verknüpft. Im Acceleo Projekt *twinCodeGenerator* werden aus den verknüpften Elementen die Files *twinComponent.txt*, *twinComponent.cs* und *SM Code Files* erstellt. Das twinComponents.cs File wird von der *twinGenerator API* verwendet, um im twin Projekt *twinSimulation_DF* Simulationskomponenten und Verknüpfungen zu erzeugen. Diese werden mit Informationen aus den restlichen generierten Files ergänzt. Im weiteren Verlauf des Kapitels wird der beschriebene Ablauf anhand des autonomen Transportroboters Servus ARC3, welcher Bestandteil der Digital Factory ist, präsentiert.

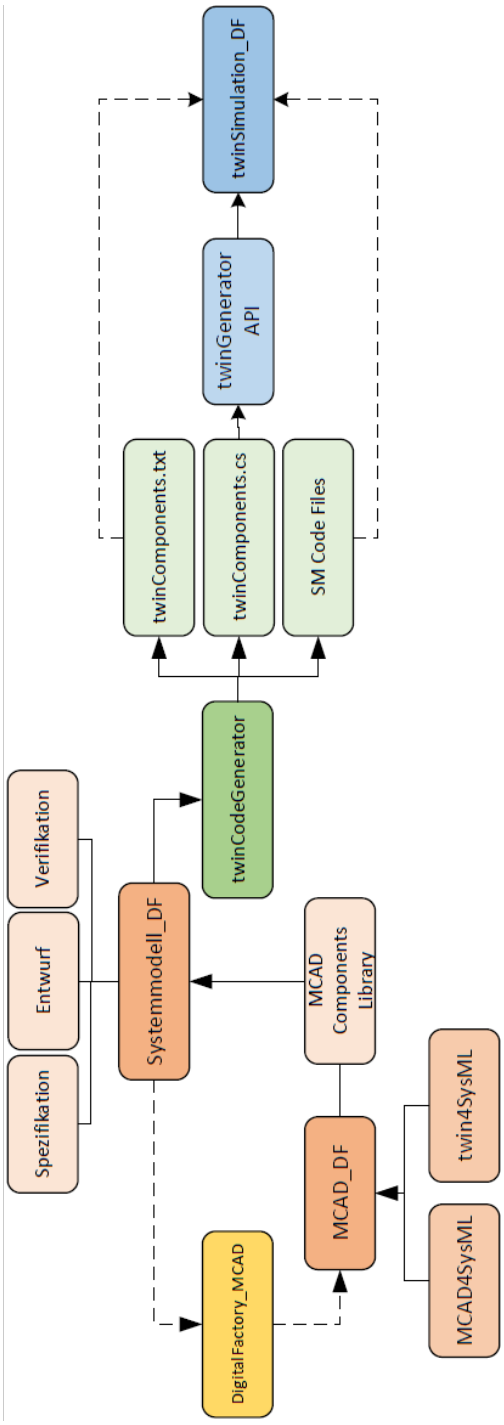


Abbildung 6.1: Gesamter Ablauf im Überblick

6.1 Servus Transportroboter

Es wird nun das Ergebnis am Beispiel des Servus Transportroboters präsentiert. Es wird dabei gezeigt, wie Informationen in den SysML Modellen modelliert und mit Modelltransformation in das Simulationsmodell transformiert werden. Das Resultat ist ein Simulation Components Diagram in der twin-Simulationssoftware, welches zur Simulation des Servus Transportroboters verwendet werden kann.

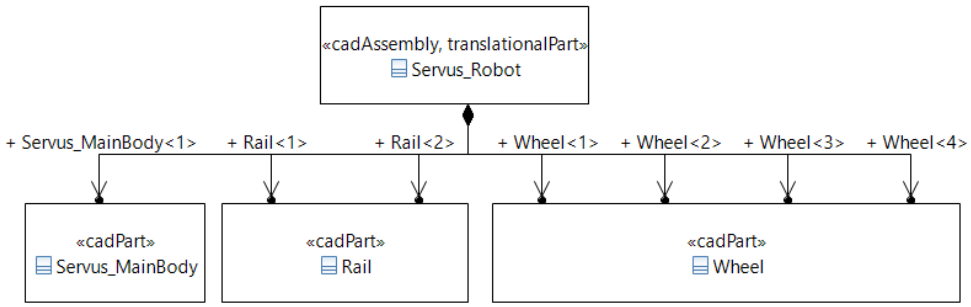


Abbildung 6.2: Servus Transportroboter im Modell *MCAD_DF*

In Abbildung 6.2 wird die Modellierung des Transportroboters Servus ARC3 im Modell *MCAD_DF* präsentiert. Er wurde mit dem Block *Servus_Robot* modelliert und besitzt die Stereotypen *cadAssembly* (MCAD4SysML Profile) und *translationalPart* (twin4SysML Profile).

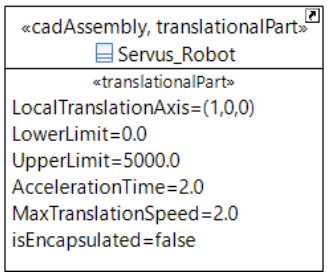


Abbildung 6.3: Servus_Robot Tagged Values

Bei der Anwendung des Stereotyps *translationalPart* werden Tagged Values initialisiert. In Abbildung 6.3 werden diese präsentiert. Sie werden während der Transformation in die resultierenden Simulationskomponenten übertragen.

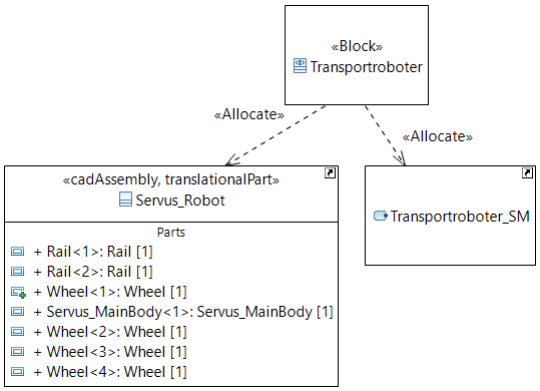


Abbildung 6.4: Servus Transportroboter im Modell *Systemmodell_DF*

Abbildung 6.4 zeigt die Modellierung des Blocks *Transportroboter* auf der untersten Abstraktionsebene *Components Level* des SysML Modells *Systemmodell_DF*. Der Block *Servus_Robot* und die SM *Transportroboter_SM*, welche das Verhalten des Transportroboters beschreibt, werden durch die *Allocate* Beziehung mit dem Transportroboter verknüpft.

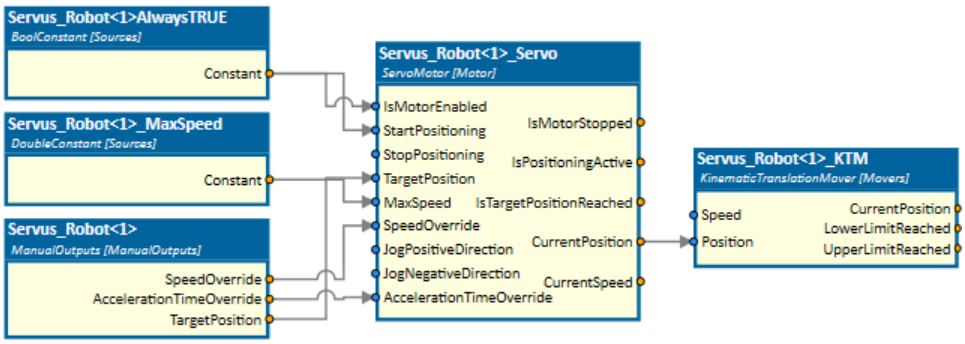


Abbildung 6.5: Generierte Simulationskomponenten für den Stereotyp *translationalPart*

Nach Anwendung des Profiles *twin4SysML* und der Modellierung im Modell Systemmodell_DF, wird die M2T Transformation mit Acceleo ausgeführt. Diese erzeugt Files, welche zur Generierung des Modells *twinSimulation_DF* verwendet werden. Der generierte Code im File *twinComponents.cs* für den Block *Servus_Robot* erstellt die Simulationskomponenten und Verknüpfungen in twinSimulation_DF, welche in Abbildung 6.5 abgebildet sind. Mit der Simulationskomponente *KinematicTranslationMover* wird erreicht, dass das 3D Objekt bewegt werden kann. Durch die Simulationskomponente *ServoMotor* kann die

Position aufgrund von Beschleunigung, Geschwindigkeit und Zielpostion berechnet werden. Mit der Simulationskomponente *ManualOutputs* können Beschleunigung, Geschwindigkeit und Zielposition während der Simulation angepasst werden. Die Simulationskomponente *BoolConstant* wird verwendet, um die Eingänge *IsMotorEnabled* und *StartPositioning* der Simulationskomponente ServoMotor auf *true* zu setzten. Dadurch wird bei einer Änderung der Zielposition sofort mit der Bewegung begonnen. Die Simulationskomponente *DoubleConstant* setzt den Eingang *MaxSpeed* des ServoMotor auf den Wert des Tagged Value im Block Servus_Robot.

```
23  ----------------------------------------------------------------------------------
24  Information to the Components of translationalPart Servus_Robot<1>
25
26  TODO:
27  1) Set Rigit Body Behaviour of 3D Object Servus_Robot<1> to "Kinematic"
28  2) Set following Parameters of the Simulation Components:
29      => Component Servus_Robot<1>_KTM
30          Object3D = Servus_Robot<1>
31  3) Start Simulation and move Part with Parameters of Component Servus_Robot<1>_MOUT
32  ----------------------------------------------------------------------------------
```

Abbildung 6.6: Zusatzinformationen der Simulationskomponenten für den Stereotyp *translationalPart*

Damit sich der Servus Transportroboter in der Simulation bewegt, müssen noch zusätzliche Einstellungen getroffen werden, welche nicht direkt durch Transformation und API möglich waren. Diese werden im File *twinComponents.txt* aufgelistet und in Abbildung 6.6 angezeigt.

```
/*
Script Component declaration for the State Machine "Transportroboter_SM"
*/
    var c162_SCR = testEnvironment.CreateComponent<V3S.ComponentLibrary.Basic.Automation.ScriptComponent>();
    componentContainer.Add(c162_SCR);
    c162_SCR.ScriptLanguage = V3S.ComponentLibrary.Basic.Automation.ScriptLanguage.CSharp;
    c162_SCR.RunningType = RunningType = V3S.ComponentLibrary.Basic.Automation.ScriptRunningType.BeforeSimulationStep;
    c162_SCR.Name = "Transportroboter_SM_SCR";
    Variable state = new Variable() {Name = "state",Type = V3S.ComponentLibrary.Basic.Automation.VariableType.Integer,InitialValue="0"};
    Variable time = new Variable() {Name = "time", Type = V3S.ComponentLibrary.Basic.Automation.VariableType.Double, InitialValue = "0.0"};
    c162_SCR.AddVariable(state);
    c162_SCR.AddVariable(time);
    c162_SCR.ScriptText = @"copy ScriptText from File =>Transportroboter_SM_ScriptText.cs";
```

Abbildung 6.7: Script im File *twinComponents.cs*

Für die SM *Transportroboter_SM* wird der Code zur Erzeugung der Simulationskomponente *ScriptComponent* im File *twinComponents.cs* und der Code für das Script selbst im File *Transportroboter_SM_ScriptText.cs* generiert. In Abbildung 6.7 wird der Code im File *twinComponents.cs* abgebildet. Es werden die Parameter *ScriptLanguage*, *RunningType* und *Name* gesetzt. Außerdem

werden die Variablen *state* und *time* definiert und hinzugefügt. Im Parameter *ScriptText* wird auf das File *Transportroboter_SM_ScriptText.cs* verwiesen.

Transportroboter_SM_ScriptText.cs

```
1  /*
2  ********************************************************************************************
3      This is a automatic generated Script from the UML State Machine
4          => Transportroboter_SM <=
5  ********************************************************************************************
6  */
7
8  //Start of user code Inputs
9      //TODO: Link Inputs to Change Event Variables
10     //z.B. var start = Inputs.Get<bool>(0);
11 //End of user code
12
13 time += SimulationWorld.SimulationTimeStep;
14
15 //States
16 enum States
17 {
18 IDLE,
19 MOVE2PICK,
20 MOVE2PLACE,
21 PICK,
22 PLACE
23 }
24
25 //ChangeEvents
26 var ev_Start_Job = ((Start_Job == true)&&(time>SimulationWorld.SimulationTimeStep))? true:false;
27 var ev_PickPosReached = (((targetReached == true))&&(time>SimulationWorld.SimulationTimeStep))? true:false;
28 var ev_Ready2Move = ((time >= 4.0)&&(time>SimulationWorld.SimulationTimeStep))? true:false;
29 var ev_PlacePosReached = (((targetReached== true))&&(time>SimulationWorld.SimulationTimeStep))? true:false;
30
31 switch(state)
32 {
33     case (int)States.IDLE:
34         //Start of user code IDLE
35             //TODO: Code for State IDLE should be implemented
36         //End of user code
37
38         if (ev_Start_Job)
39         {
40             state = (int)States.MOVE2PICK;
41             time = 0.0;
42         }
43
44         break;
```

Abbildung 6.8: Generierter SM-Code

Ein Teil dieses Files wird in Abbildung 6.8 abgebildet. In den Zeilen 8-11 befindet sich der geschützte Bereich für die Verknüpfung der Inputs mit Variablen im Code. In der Zeile 13 wird die Variable *time* um die Zeit eines Simulationsschritts erhöht. In den Zeilen 16-23 befindet sich die Enumeration für die States und in den Zeilen 25-29 die Variablen für die Change Events. In Zeile 31 beginnt die switch-Anweisung mit einem *case* für jeden State. Diese beinhalten wiederum einen geschützen Bereich für User Code (Zeile 34-36) und eine Transition, welche durch die Change Event Variablen ausgelöst wird (Zeile 38-42).

```
52  ----------------------------------------------------------------------------------------
53  Information to the Script Component "Transportroboter_SM_SCR"
54      TODO:
55      1) Define Inputs and Outputs of the Script Component
56      2) Copy Code from the File Transportroboter_SM_ScriptText.cs into Parameter ScriptText
57      3) Link Inputs to Variables in the "user code Inputs" Section
58      4) Set Outputs in the "user code STATE" Sections of the switch-Statement
59      5) Link Inputs and Outputs in the Simulation Components Diagram with other Simulation Components
60  ----------------------------------------------------------------------------------------
```

Abbildung 6.9: Zusatzinformation für die Script-Simulationskomponente

Um das Script in der Simulation verwenden zu können, müssen zusätzliche Einstellungen getroffen werden. Diese wurden wiederum im File *twinComponents.txt* dokumentiert und in Abbildung 6.9 angezeigt.

```
8   //Start of user code Inputs
9       //TODO: Link Inputs to Change Event Variables
10      //z.B. var start = Inputs.Get<bool>(0);
11      var Start_Job = Inputs.Get<bool>(0);
12      var targetReached = Inputs.Get<bool>(1);
13      var PickPosition = Inputs.Get<double>(2);
14      var PlacePosition = Inputs.Get<double>(3);
15  //End of user code
```

Abbildung 6.10: Verknüpfung Inputs und Variablen im User Code

Diese Schritte wurde folgendermaßen umgesetzt: Es wurden die Inputs *Start_Job*, *IsPositionReached*, *PickPosition* und *PlacePosition*, sowie die Outputs *TargetPosition*, *PickPart* und *PlacePart* erstellt. Der Inhalt des entsprechenden Files in den Parameter *ScriptText* kopiert, und die Inputs mit Variablen verknüpft(Abbildung 6.10).

```
//Start of user code MOVE2PICK
    //TODO: Code for State MOVE2PICK should be implemented
    Outputs[0] = PickPosition;
//End of user code
```

Abbildung 6.11: Setzen des Outputs im User Code

An den dafür vorgesehenen Stellen im Code wurden, wie in Abbildung 6.11, die Outputs gesetzt und am Ende die Verknüpfungen zu anderen Simulationskomponenten erstellt.

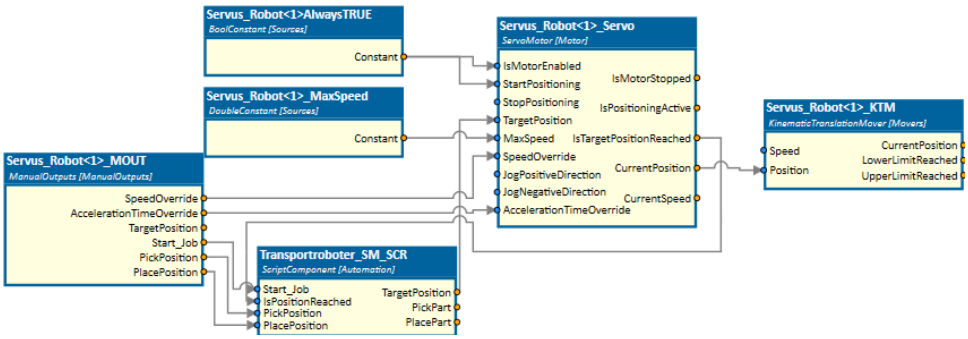


Abbildung 6.12: Servus_Robot und Transportroboter_SM im twinSimulation_DF

Das resultierende Ergebnis im Modell *twinSimulation_ DF* wird in Abbildung 6.12 abgebildet. Der Simulationskomponente *ManualOutputs* wurden Outputs hinzugefügt und mit den Inputs *Start_ Job*, *PickPosition* und *PlacePosition* verbunden. Der Input *IsPositionReached* wird mit dem Output *IsTargetPosition-Reached* der Simulationskomponente *ServoMotor* verbunden. Der Output TargetPosition wird mit dem Gleichnamigen Input der Komponente ServoMotor verbunden. Es können somit Positionen definiert werden. Mit dem Signal Start_Job werden diese Positionen nacheinander angefahren und die Outputs *PickPart* und *PlacePart* für das Laden und Entladen gesetzt.

7 Zusammenfassung & Ausblick

Die präsentierte Arbeit beschäftigt sich mit der Modellierung von komplexen Systemen unter Berücksichtigung folgender Ziele:

- Formale Beschreibung des Systems
- Rückverfolgbare Modellierung von Anforderungen
- Integrierung domänenspezifischer Modelle
- Single Source of Truth
- Validierung des Modells
- Anbindung an eine Simulationssoftware

Zur formalen Beschreibung des Systems wurde die Modellierungssprache SysML verwendet. Es wurde ein Systemmodell erstellt, welches Spezifikation, Entwurf und Verifikation des Systems enthält. Anforderungen wurden mit Artefakten des Systementwurfs verknüpft und damit Rückverfolgbar modelliert. Das Profile *MCAD4SysML* wurde erstellt, um Informationen eines MCAD Modells in ein SysML-Domänenmodell zu übernehmen, wodurch die Möglichkeit einer automatisierten Integration von MCAD Modellen mit Hilfe von M2M Transformationen geschaffen wurde. Eine solche Transformation wurde im Rahmen dieser Arbeit nicht umgesetzt. Die Informationen wurden manuell übernommen und in einer *Model Library* gespeichert, welche in das Systemmodell importiert wird. In diesem werden Mappings zwischen den Systementwurfs- und Library-Artefakten erzeugt. Durch diese Art der Zusammenführung von Informationen in ein Systemmodell entsteht eine Single Source of Truth. Mit dem Modellierungstool *Papyrus* wurde das erstellte Modell validiert. Das SysML-Systemmodell wurde in ein Simulationsmodell der Simulationssoftware *twin* (Eberle Automatische Systeme GmbH & Co KG) überführt. Dafür wurden zu Simulierende Artefakte im Modell mit Stereotypen des Profiles *twin4SysML* ergänzt und das Verhalten von Systemkomponenten mit State Machines modelliert. Mittels einer Transformation in *Acceleo* wird für diese Artefakte Code generiert. Dieser wird mit einer *API* verwendet um Elemente im Simulationsmodell zu erzeugen. Zusätzlich generierte Files helfen bei der Vervollständigung des Simulationsmodells.

Eine künftige Aufgabe wäre die Berücksichtigung weiterer Domänen, sowie die Anbindung an domänenspezifische Tools durch M2M Transformationen. Außerdem wären zusätzliche Überprüfungen, beispielsweise ob alle Anforderungen satisfied/verified sind, im Rahmen der Validierung hilfreich. Benutzerdefinierte Regeln können mit OCL definiert werden und die Validierung ergänzen. Für die Anbindung an die Simulationssoftware könnten twin4SysML Profile und Transformation mit weiteren Stereotypen ergänzt werden. Ein Grund, warum für die Transformation Acceleo verwendet wird, ist die einfache Integration mit Papyrus. Bei komplexen Transformationen wird der Code jedoch unübersichtlich und eine alternative Transformation mittels Xtend wäre in Erwägung zu ziehen.

Literatur

[1] Oliver Alt. *Car Multimedia Systeme Modell-basiert Testen mit SysML*. Springer-Verlag, 25. März 2009. 208 S. ISBN: 978-3-8348-0761-8.

[2] Oliver Alt. *Modellbasierte Systementwicklung mit SysML*. Google-Books-ID: rS1QAgAAQBAJ. Carl Hanser Verlag GmbH Co KG, 1. März 2012. 226 S. ISBN: 978-3-446-43127-0.

[3] *ATL | The Eclipse Foundation*. URL: https://www.eclipse.org/atl/ (besucht am 17.05.2021).

[4] *ATL/Tutorials - Create a simple ATL transformation - Eclipsepedia*. URL: https://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation#The_metamodels (besucht am 24.08.2021).

[5] *AutomationML - story*. Automationml.org. URL: http://www.automationml.org/o.red.c/story.html (besucht am 27.05.2021).

[6] G. Barbieri u.a. „A SysML based design pattern for the high-level development of mechatronic systems to enhance re-usability". In: *IFAC Proceedings Volumes*. 19th IFAC World Congress 47.3 (1. Jan. 2014), S. 3431–3437. ISSN: 1474-6670. DOI: 10.3182/20140824-6-ZA-1003.00615. URL: https://www.sciencedirect.com/science/article/pii/S1474667016421361 (besucht am 09.03.2021).

[7] L. Bassi u.a. „A SysML-Based Methodology for Manufacturing Machinery Modeling and Design". In: *IEEE/ASME Transactions on Mechatronics* 16.6 (Dez. 2011). Conference Name: IEEE/ASME Transactions on Mechatronics, S. 1049–1062. ISSN: 1941-014X. DOI: 10.1109/TMECH.2010.2073480.

[8] L. Berardinelli u.a. *Cross-disciplinary engineering with AutomationML and SysML*. 2016. DOI: 10.1515/auto-2015-0076.

[9] Luca Berardinelli u.a. „Model-Driven Systems Engineering: Principles and Application in the CPPS Domain". In: *Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects*. Journal Abbreviation: Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engi-

neering Projects. 7. Mai 2017, S. 261–299. ISBN: 978-3-319-56344-2. DOI: 10.1007/978-3-319-56345-9_11.

[10] Ulrich Dahmen und Juergen Rossmann. „Experimentable Digital Twins for a Modeling and Simulation-based Engineering Approach“. In: *2018 IEEE International Systems Engineering Symposium (ISSE)*. 2018 IEEE International Systems Engineering Symposium (ISSE). Okt. 2018, S. 1–8. DOI: 10.1109/SysEng.2018.8544383.

[11] *Digital Factory Vorarlberg*. URL: https://www.fhv.at/forschung/digital-factory-vorarlberg/ (besucht am 12. 05. 2021).

[12] M. Eigner u. a. *An approach for a model based development process of cybertronic systems*. 2014. URL: https://www.semanticscholar.org/paper/An-approach-for-a-model-based-development-process-Eigner-Muggeo/205ca9b798c2e6db57c524c1d908ba2ce363dc73 (besucht am 25. 05. 2021).

[13] Martin Eigner, Walter Koch und Christian Muggeo, Hrsg. *Modellbasierter Entwicklungsprozess cybertronischer Systeme: Der PLM-unterstützte Referenzentwicklungsprozess für Produkte und Produktionssysteme*. Springer Vieweg, 2017. ISBN: 978-3-662-55123-3. DOI: 10.1007/978-3-662-55124-0. URL: https://www.springer.com/de/book/9783662551233 (besucht am 25. 05. 2021).

[14] Eclipse Foundation. *Acceleo/User Guide - Eclipsepedia*. wiki.eclipse.org. URL: https://wiki.eclipse.org/Acceleo/User_Guide (besucht am 05. 08. 2021).

[15] Sanford Friedenthal, Alan Moore und Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Elsevier, 31. Okt. 2011. 642 S. ISBN: 978-0-12-385206-9.

[16] *INCOSE*. URL: https://www.incose.org/incose-impact (besucht am 14. 05. 2021).

[17] Technical Operations INCOSE, Hrsg. *INCOSE Systems Engineering Vision 2020*. Sep. 2007. URL: http://www.ccose.org/media/upload/SEVision2020_20071003_v2_03.pdf (besucht am 18. 05. 2021).

[18] Uwe Kaufmann und Michael Pfenning. „Was der Maschinenbau von der Softwareentwicklung lernen kann - durchgängige Integration disziplinspezifischer Modelle durch den Einsatz von Modellierungssprachen“. In: (16. Nov. 2014). URL: https://www.researchgate.net/publication/326416303_Was_der_Maschinenbau_von_der_Softwareentwicklung_lernen_kann_-_durchgangige_Integration_disziplinspezifischer_Modelle_durch_den_Einsatz_von_Modellierungssprachen.

[19] Konstantin Kernschmidt, Stefan Feldmann und Birgit Vogel-Heuser. „A model-based framework for increasing the interdisciplinary design of mechatronic production systems“. In: *Journal of Engineering Design* 29 (2. Nov. 2018), S. 617–643. DOI: 10.1080/09544828.2018.1520205.

[20] Konstantin Kernschmidt und Birgit Vogel-Heuser. „An interdisciplinary SysML based modeling approach for analyzing change influences in production plants to support the engineering“. In: *2013 IEEE International Conference on Automation Science and Engineering (CASE)*. 2013 IEEE International Conference on Automation Science and Engineering (CASE). ISSN: 2161-8089. Aug. 2013, S. 1113–1118. DOI: 10.1109/CoASE.2013.6654030.

[21] H. Li u.a. „Consistent Automated Production Systems Modeling in a Multi-disciplinary Engineering Workflow“. In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society. ISSN: 2577-1647. Okt. 2018, S. 2971–2978. DOI: 10.1109/IECON.2018.8591439.

[22] A. Lüder, L. Hundt und A. Keibel. „Description of manufacturing processes using AutomationML“. In: *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*. 2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010). ISSN: 1946-0759. Sep. 2010, S. 1–8. DOI: 10.1109/ETFA.2010.5641346.

[23] Mathworks. *Was ist ein Digital Twin?* 21. Mai 2021. URL: https://de.mathworks.com/discovery/digital-twin.html (besucht am 21.05.2021).

[24] Christian Muggeo und Michael Pfenning. *Die Rolle von MBSE und PLM im Industrial Internet*. 6. Nov. 2015. URL: https://www.researchgate.net/profile/Michael-Pfenning-2/publication/326416882_Die_Rolle_von_MBSE_und_PLM_im_Industrial_Internet/links/5b4c88300f7e9b4637de2c59/Die-Rolle-von-MBSE-und-PLM-im-Industrial-Internet.pdf.

[25] *OMG SysML v. 1.6 [Dec 2019]*. Dez. 2019. URL: https://www.omg.org/spec/SysML/1.6/ (besucht am 02.08.2021).

[26] Jan Peleska. *Specification of Embedded Systems Summer Semester 2020 Session 2 Modelling Interfaces and Structure*. 4. Mai 2020. URL: http://www.informatik.uni-bremen.de/agbs/jp/jp_papers_e.html (besucht am 01.06.2021).

[27] Ahsan Qamar. *An Integrated Approach towards Model-Based Mechatronic Design*. Publisher: KTH Royal Institute of Technology. 2011. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-35374 (besucht am 04.06.2021).

[28] Prof. Dr. Günther Schuh. *PLM (Product Lifecycle Management) — Enzyklopaedie der Wirtschaftsinformatik*. Enzyklopedia der Wirtschaftsinformatik. 25. März 2020. URL: https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/informationssysteme/Sektorspezifische-Anwendungssysteme/Product-Life-Cycle-Management/index.html (besucht am 26.05.2021).

[29] Aditya Shah, Dirk Schaefer und Christiaan Paredis. „Enabling multi-view modeling with SysML profiles and model transformations“. In: *IFAC Proceedings Volumes Volume 47, Issue 3, 2014, Pages 3431-3437*. 19th IFAC World Congress. Cape Town, South Africa, Aug. 2014. DOI: https://doi.org/10.3182/20140824-6-ZA-1003.00615.

[30] Rainer Stark und Thomas Damerau. „Digital Twin“. In: *CIRP Encyclopedia of Production Engineering*. Hrsg. von Sami Chatti und Tullio Tolio. Berlin, Heidelberg: Springer, 2019, S. 1–8. ISBN: 978-3-642-35950-7. DOI: 10.1007/978-3-642-35950-7_16870-1. URL: https://doi.org/10.1007/978-3-642-35950-7_16870-1 (besucht am 20.05.2021).

[31] Kleanthis Thramboulidis. „Model-Integrated Mechatronics—Toward a New Paradigm in the Development of Manufacturing Systems“. In: *Industrial Informatics, IEEE Transactions on* 1 (1. März 2005), S. 54–61. DOI: 10.1109/TII.2005.844427.

[32] Kleanthis Thramboulidis. „The 3+1 SysML View-Model in Model Integrated Mechatronics“. In: *JSEA* 3 (1. Jan. 2010), S. 109–118. DOI: 10.4236/jsea.2010.32014.

[33] Tim Weilkiens. *Systems Engineering mit SysML/UML: Anforderungen, Analyse, Architektur. Mit einem Geleitwort von Richard Mark Soley*. dpunkt.verlag, 3. Sep. 2014. 517 S. ISBN: 978-3-86491-543-7.

[34] Thomas Westermann. „Einführung“. In: *Modellbildung und Simulation: Mit einer Einführung in ANSYS*. Hrsg. von Thomas Westermann. Berlin, Heidelberg: Springer, 2021, S. 1–12. ISBN: 978-3-662-63045-7. DOI: 10.1007/978-3-662-63045-7_1. URL: https://doi.org/10.1007/978-3-662-63045-7_1 (besucht am 19.05.2021).

[35] *What is SysML? | OMG SysML*. URL: https://www.omgsysml.org/what-is-sysml.htm (besucht am 14.05.2021).

[36] *Xtend - Documentation*. URL: https://www.eclipse.org/xtend/documentation/index.html (besucht am 27. 05. 2021).

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dornbirn, am 06.09.2021 Sebastian Franz