# T U M

I N S T I T U T F Ü R I N F O R M A T I K

# AutoVIBN – Abschlussbericht: Automatische Generierung von Verhaltensmodellen aus CAD-Daten für die qualitätsorientierte virtuelle Inbetriebnahme

Jewgenij Botaschanjan, Thomas Hensel, Benjamin Hummel, Alexander Lindworsky, Michael F. Zäh, Gunther Reinhart, Manfred Broy

TUM-I1012
Juni 10

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-06-I1012-0/1.-FI


Druck: Institut für Informatik der
Technischen Universität München

**AutoVIBN – Abschlussbericht**

Automatische Generierung von Verhaltensmodellen
aus CAD-Daten für die qualitätsorientierte virtuelle Inbetriebnahme

4. Juni 2010

Das in diesem Abschlussbereicht beschriebene Forschungsvorhaben (Forschungsvorhabensnummer: ZN274) wurde aus Haushaltsmitteln des Bundesministeriums für Wirtschaft und Technologie (BMWI) über die Arbeitsgemeinschaft industrieller Forschungsvereinigungen „Otto von Guericke“ e.V. (AiF) gefördert.

Dr. Jewgenij Botaschanjan
Thomas Hensel
Benjamin Hummel
Alexander Lindworsky

Technische Universität München
Institut für Werkzeugmaschinen und Betriebswissenschaften
Prof. Dr.-Ing. Michael F. Zäh
Prof. Dr.-Ing. Gunther Reinhart

Technische Universität München
Institut für Informatik
Software & Systems Engineering
Prof. Dr. Dr. h.c. Manfred Broy

# Zusammenfassung der Projektergebnisse

Die zwei nachfolgend beschriebenen Entwicklungen im Bereich des Maschinen- und Anlagenbaus dienten als Motivation für das Forschungsvorhaben AutoVIBN:

- Die Funktionalität moderner Produktionssysteme wird zunehmend durch die Steuerungssoftware realisiert. Zudem steigt durch neue technische Möglichkeiten und Kundenwünsche der Funktionsumfang der Maschinen. Bedingt durch die starke Kopplung der Steuerung an das Produktionssystem ist allerdings ein isolierter Test der Steuerungssoftware nur sehr eingeschränkt möglich. Aus diesem Grunde findet ein großer Teil der Testaktivitäten meist in der Integrations- und Inbetriebnahmephase statt. Ein Ansatz, die Software frühzeitig zu prüfen, ist die virtuelle Inbetriebnahme (im Folgenden VIBN genannt). Hierbei wird die reale Anlage durch ein virtuelles Maschinenmodell ersetzt. Dadurch ist es möglich, den Gutablauf des Programms sowie Fehlerszenarien schon während der Konstruktionsphase zu durchlaufen und aufwandsarm zu prüfen. Der Aufbau dieser Simulationsmodelle gestaltet sich derzeit noch sehr aufwändig, weshalb die VIBN im industriellen Umfeld als Methode der Qualitätssicherung noch wenig Anwendung erfahren hat.
- Moderne mechatronische Produktionssysteme sind durch ein Zusammenwirken der Disziplinen Mechanik, Elektrotechnik und Software gekennzeichnet. Vor allem in der Entwicklungsphase stellt die Integration der unterschiedlichen Fachbereiche eine Herausforderung für die Maschinenhersteller dar. Sequenziell aufgebaute Prozesse führen zu langen Entwicklungszeiten. Zusätzlich erschweren die mangelnde Dokumentation und die unzureichend unterstützte Kommunikation die Zusammenarbeit zwischen den Abteilungen.

Im Rahmen des Forschungsvorhabens AutoVIBN wurden diese Problemstellungen aufgegriffen und eine Beschreibungstechnik entwickelt, die eine disziplinunabhängige, funktionale Modellierung einer Anlage erlaubt. Die Modellierungstechnik, im weiteren Verlauf des Dokuments *Funktionsbeschreibung* genannt, wurde durch ein hier entwickeltes Softwarewerkzeug (AF/ STEM) ergänzt. Es handelt sich dabei jedoch lediglich um ein Funktionsmuster und nicht um ein serienreifes Produkt. Dieses ermöglicht die Erstellung und die frühzeitige Simulation von Funktionsbeschreibungen sowie die Nutzung der gesammelten Informationen für die Anlagenentwicklung.

Des Weiteren ist für eine bessere Integration mit disziplinspezifischen Modellen eine werkzeuggestützte Rückführung und Konsistenzprüfung von CAD-Daten in die Funktionsbeschreibung umgesetzt. Zusätzlich wurden Richtlinien bezüglich der CAD-Modellierung erarbeitet, die einen maximalen Automatisierungsgrad für die nachfolgende Modellgenerierung der VIBN sicherstellen sollen.

Für die Anwendung im Bereich der Validierung mechatronischer Systeme wurde im Projektvorhaben AutoVIBN zudem eine Technik zur orthogonalen Fehlermodellierung für Hardware-Komponenten entworfen und in der Funktionsbeschreibung implementiert. Dieser Ansatz erlaubt es, das Fehlverhalten von Systembestandteilen zu spezifizieren und die Auswirkungen auf das Gesamtverhalten zu beobachten. Zudem wurden randomisierte Testgenerierungsverfahren konzipiert und in das Funktionsmuster integriert. Damit lassen sich automatisiert Testfälle aus den Modellen ableiten.

Die Funktionsbeschreibung bildet zudem die Grundlage zur Generierung des VIBN-Modells. Dieses wird um entsprechende Informationen erweitert und als C++-Code direkt ausgeleitet. Der Simulationskern kann anschließend mit Hilfe spezieller Hardware an die reale Steuerung gekoppelt werden.

Ausgehend von den oben genannten Aspekten wurde eine Methode erarbeitet, die

- eine interdisziplinäre Entwicklung von mechatronischen Systemen unterstützt und
- die eine Modellerstellung für die VIBN von Produktionssystemen weitestgehend automatisiert.

Hierfür wurden Daten genutzt, die im Entwicklungsprozess bereits vorhanden sind. Die entwickelten Techniken und Methoden wurden an zwei industriellen Fallstudien erfolgreich erprobt.

Zusammenfassend kann festgehalten werden, dass die Ziele des Forschungsprojektes AutoVIBN erreicht wurden.

# Inhaltsverzeichnis

# 1 Zielsetzung

## 1.1 Forschungsthema

Automatische Generierung von Verhaltensmodellen aus CAD-Daten für die qualitätsorientierte virtuelle Inbetriebnahme (AutoVIBN).

## 1.2 Problemstellung

Eine Möglichkeit zur frühzeitigen Prüfung der Steuerungssoftware ist die virtuelle Inbetriebnahme (VIBN). Hierbei wird die reale Maschine durch ein virtuelles Modell ersetzt und an die Steuerung gekoppelt. Dadurch kann die Steuerungssoftware ohne die physische Präsenz der Anlage getestet werden. Mit einem derartigen Ansatz wird die Softwareentwicklung parallel zu Konstruktion, Fertigung und Montage ermöglicht und reduziert somit den Zeitdruck bei der realen Inbetriebnahme. Durch die Nutzung der Simulationsmodelle lässt sich die Steuerungssoftware iterativ und inkrementell entwickeln. Dies erlaubt neben dem frühzeitigen Erkennen von Fehlern im Programmcode auch das rechtzeitige Aufdecken von Optimierungspotenzialen der Maschinen- und Anlagenabläufe sowie eine stärker verzahnte Entwicklung von Mechanik, Elektrotechnik und Software. Derzeit gestaltet sich der Modellerstellungsprozess für die VIBN noch sehr zeitintensiv, da dieser einen vornehmlich manuellen Charakter aufweist.

## 1.3 Forschungsziel

Das Ziel des Forschungsprojektes AutoVIBN ist die automatische Generierung physikalischer Verhaltensmodelle für die VIBN. Die Basis bildet hierbei eine sog. Funktionsbeschreibung, die bereits in der Projektierungs- und Planungsphase eingesetzt werden kann. Das abstrakte Maschinenmodell in der Funktionsbeschreibung lässt sich wesentlich einfacher und schneller erstellen, als dies beim VIBN-Modell der Fall ist. Durch eine Integration der im Entwicklungsprozess vorhandenen Informationen, hier der CAD-Daten, kann das übergeordnete Modell erweitert werden. Das für den Steuerungstest notwendige VIBN-Modell soll letztendlich vollständig aus dem Funktionsmodell abgeleitet werden. Der Nutzen dieser Vorgehensweise besteht darin, dass die Simulation von Beginn an in den Entwicklungsprozess integriert werden kann.

Neben der Ableitung von VIBN-Modellen dient die Funktionsbeschreibung einem weiteren Zweck. Um deren Erstellungsaufwand zusätzlich zu amortisieren, soll diese als Abstimmungs- und Kommunikationsmittel zwischen den beteiligten Disziplinen herangezogen werden können.

# 2 Erreichte Ergebnisse

## 2.1 Zusammenfassung der Ergebnisse der Arbeitspakete

Die erzielten Ergebnisse werden anhand der Arbeitspakete (AP) strukturiert, diese Einteilung folgt dem bewilligten Projektantrag. Im vorliegenden Abschnitt wird lediglich eine knappe Zusammenfassung der Resultate vermittelt. Die detaillierten Erläuterungen sind in Kapitel 3 dargestellt.

### 2.1.1 AP1: Anforderungsanalyse bei den Projektpartnern

Ausgehend von der Ist-Situation der am Projekt beteiligten Unternehmen wurde untersucht, wie die VIBN als möglicher Bestandteil eines Standardvorgehens im Rahmen der Qualitätssicherung der Steuerungssoftware in den betrieblichen Gesamtablauf eingebunden werden kann. Hierzu wurden einerseits die jeweiligen Entwicklungsprozesse analysiert und verglichen. Andererseits wurden die zu simulierenden Versuchsträger und die genutzten CAD-Systeme identifiziert.

### 2.1.2 AP2: Funktionsbeschreibung für die Maschinenmodellierung

Da die Spezifikation und die Modellierung von Maschinenfunktionen in frühen Entwicklungsphasen eine Beschreibung der Funktion darstellt, wird im Folgenden der Begriff der Funktionsbeschreibung verwendet. Zur Spezifikation des Verhaltens wurden bekannte Methoden (z. B. Zustandsgraphen) genutzt, die sowohl den Mechanik- und Elektrokonstrukteuren als auch den Steuerungstechnikern bekannt sind. Besondere Anforderungen an die Funktionsbeschreibung, bspw. die Abbildung von Maschinenverhalten, wurden in diesem Arbeitspaket festgelegt. Mangels geeigneter systemtechnischer Unterstützung wurde ein Funktionsmuster eines Softwarewerkzeuges (AF/STEM) erarbeitet, das zur Erstellung, zur Bearbeitung und zur Simulation diente. Dieses wurde im Fortlauf des Projektes stetig erweitert (vgl. Abbildung 2.1).

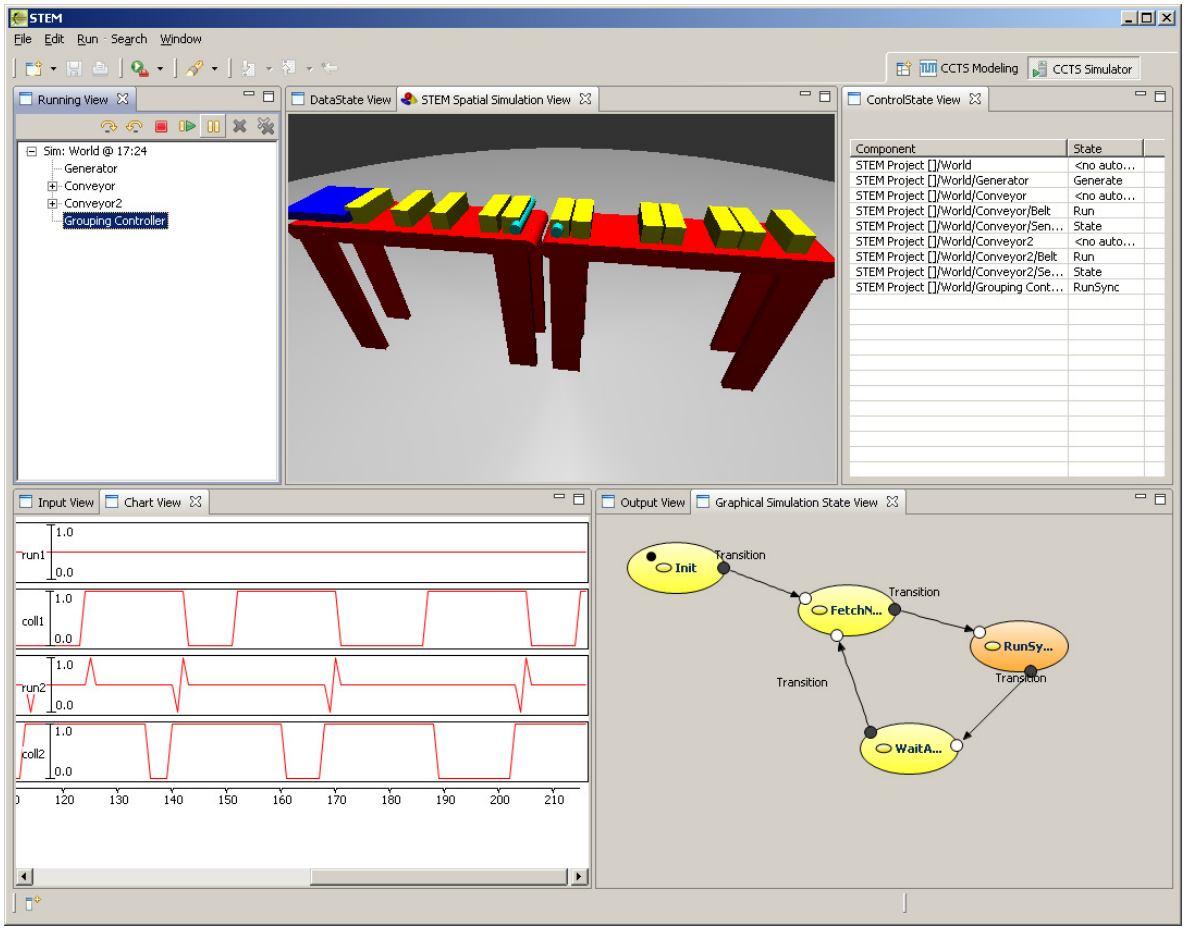



Abbildung 2.1: Das Werkzeug AF/STEM für die Erstellung der Funktionsbeschreibung

### 2.1.3 AP3: Identifikation qualitätsrelevanter Modellmerkmale und geeigneter Konsistenzbedingungen

In diesem Schritt wurden Konsistenzbedingungen für die benutzten Artefakte definiert, mit denen sich die Qualität und die Verständlichkeit der Modelle zu jedem Zeitpunkt der Entwicklung messen ließen. Der Schwerpunkt lag hierbei auf der Konsistenz zwischen der Funktionsbeschreibung und den CAD-Modellen. Das Werkzeug AF/STEM wurde ebenso um entsprechende Funktionen zur Herstellung von Querverweisen zwischen den verschiedenen Modellkategorien (Tracing) ergänzt. Zudem wurde ein Verfahren zur Konsistenzprüfung entwickelt und in das Funktionsmuster integriert. Die hier beschriebenen Arbeiten wurden in AP8 aufgegriffen und fortgesetzt. Zur Unterstützung des Tests der Steuerungssoftware wurde weiterhin untersucht, wie sich im Funktionsmodell aus AP2 mögliches Störverhalten hinterlegen lässt, bspw. der Ausfall einzelner Aktoren oder Sensoren. Diese Informationen können für die automatische Erzeugung von Testfällen und weiterführenden Analysen, wie bspw. für die Fehler-Möglichkeits- und Einfluss-Analyse (FMEA) oder für die Risikoabschätzung, herangezogen werden. Eine Modellierung von Störszenarien mittels Fehlermodi und Filtern wurde erarbeitet und im Werkzeug implementiert.

### 2.1.4 AP4: Auswertung der Datenstruktur der verwendeten CAD-Systeme

Auf der Basis der in AP1 ausgewählten CAD-Werkzeuge erfolgte in diesem Abschnitt die detaillierte Untersuchung der vorhandenen Schnittstellen und der zur Verfügung stehenden Daten aus diesen Systemen. Hierzu wurde sowohl die Struktur der Maschine als auch die für VIBN relevanten Parameter extrahiert und in Form eines systemunabhängigen, maschinenlesbaren Dokuments (XML) bereitgestellt. Entsprechende Programme wurden exemplarisch jeweils für ein ausgewähltes Mechanik- und Elektro-CAD-System (im Folgenden MCAD und ECAD genannt) realisiert.

### 2.1.5 AP5: Automatische Generierung des Verhaltensmodells

Die Funktionsbeschreibung bildete die Basis für die Ableitung des VIBN-Modells. Zunächst erfolgte eine Anreicherung mit Hilfe der CAD-Daten. Des Weiteren wurde ein Modell zur Beschreibung der Schnittstellen zwischen der Funktionsbeschreibung und den realen Steuerungsein- und -ausgängen definiert, welches in den Generierungsprozess mit eingebunden wurde. Der Simulationskern konnte anschließend als C++-Programm ausgeleitet werden.

### 2.1.6 AP6: Modellintegration in die Simulationsumgebung

AP6 wurde parallel zu AP5 bearbeitet. Der unter AP5 automatisiert erstellte Simulationskern der Maschine wurde in die Simulationsumgebung integriert (vgl. Abbildung 2.2).

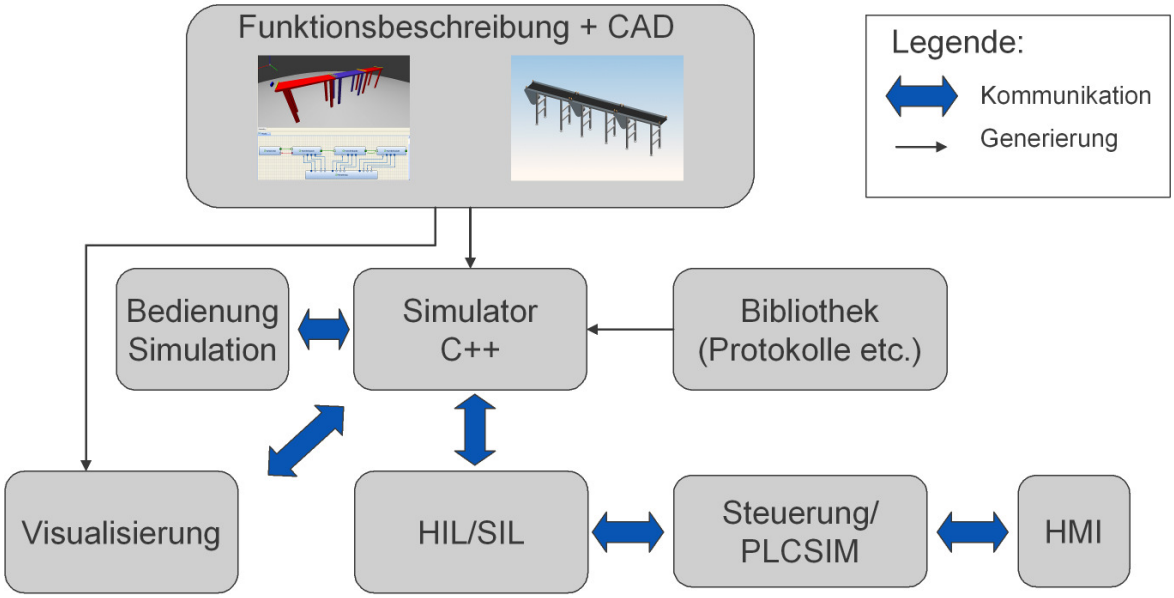



Abbildung 2.2: Struktur der Simulationsumgebung

### 2.1.7 AP7: Kopplung des Gesamtmodells an eine reale Steuerung und 3D-Visualisierung

Das als C++-Code ausgeleitete VIBN-Modell wurde im Rahmen dieses Arbeitspaketes an die reale Steuerung gekoppelt. Als Grundlage dafür wurde eine Hardware-Simulationskarte verwendet, die über Profibusanschlüsse verfügte. Der Simulator wurde um eine Anbindung dieser Karte erweitert. Für Visualisierungszwecke wurde das Werkzeug der Funktionsbeschreibung eingesetzt, welches mit dem Simulator über das Netzwerkprotokoll TCP/IP kommuniziert.

### 2.1.8 AP8: Konkretisierung der Qualitätsmerkmale und Testfallerzeugung

In diesem Arbeitspaket wurden die Qualitätsmerkmale aus AP 3 nochmals aufgegriffen und reflektiert. Der Schwerpunkt des Arbeitspakets lag jedoch auf der automatischen Erzeugung von Testfällen für die Anlage (genauer das Anlagenmodell) auf der Basis der Funktionsbeschreibung. Verschiedene, in der Literatur beschriebene Verfahren wurden auf eine Eignung in diesem Kontext untersucht. Ein auf *Random Walks* basierender Ansatz wurde schließlich für das Funktionsmodell adaptiert und zu Testzwecken realisiert.

### 2.1.9 AP9: Umsetzung der Methode anhand eines Funktionsmusters

Die hier entwickelte Methode wurde mit Hilfe von Funktionsmustern für die Funktionsbeschreibung und den VIBN-Simulator evaluiert. Zudem wurden mehrere Fallstudien durchgeführt. Als Grundlage für die ersten Versuche wurde in Abstimmung mit dem projektbegleitenden Ausschuss eine fiktive Anlage definiert, die aus Transportbändern, Greifern und Portalen bestand. Diese wurde in der Funktionsbeschreibung modelliert und anschließend als CAD-Modell aufgebaut. Zudem wurden durch den projektbegleitenden Ausschuss zwei Fallstudien real existierender Anlagen bereitgestellt, die die Erprobung unter praxisnahen Bedingungen ermöglichten.

### 2.1.10 AP10: Kritische Bewertung der erzielten Forschungsergebnisse

Die Forschungsergebnisse wurden durch die beteiligten Forschungsvereinigungen sowie den projektbegleitenden Ausschuss in regelmäßigen Statustreffen begutachtet und kritisch diskutiert.

### 2.1.11 AP11: Schlussbericht

Der vorliegende Schlussbericht wird den Zuwendungsgebern vorgelegt sowie an die Mitglieder des projektbegleitenden Ausschusses verteilt.

## 2.2 Veröffentlichungen

Dieser Abschnitt listet die im Rahmen des Projektes erschienenen Veröffentlichungen in chronologischer Reihenfolge auf. Dabei wurden sowohl Fachzeitschriften als auch Konferenzen aus dem Bereich des Maschinenbaus und der Informatik adressiert. Um eine Einordnung zu erleichtern, wird jeweils die Zusammenfassung (Abstract) des Artikels angegeben.

**Virtuelle Inbetriebnahme - Nutzenmaximierung durch Automatisierung der Modellerstellung [ZL08]** Der Beitrag beschäftigt sich mit dem Thema der Modellbildung und Simulation für den entwicklungsbegleitenden Steuerungstest. Es werden aktuelle Forschungsarbeiten des Instituts für Werkzeugmaschinen und Betriebswissenschaften (*iwb*) der Technischen Universität München vorgestellt. Dabei handelt es sich im Speziellen um das Forschungsprojekt **AutoVIBN** der Arbeitsgemeinschaft industrieller Forschungsvereinigungen e.V. (AiF), welches aus Mitteln des Bundesministeriums für Wirtschaft und Technologie (BMWi) gefördert wird. Das Projekt wird durch die Forschungsvereinigung Werkzeugmaschinen und Fertigungstechnik e.V. (FWF) und die Forschungsvereinigung Qualität e. V. (FQS) begleitet. Der beschriebene Ansatz befasst sich mit der teilautomatischen Generierung von Simulationsmodellen für die virtuelle Inbetriebnahme (VIBN). Hierbei werden vorhandene Entwicklungsdaten genutzt, die bereits modellhaft in den CAD-Systemen hinterlegt sind. Um eine entwicklungsbegleitende Vorgehensweise zu ermöglichen, wird eine formale Funktionsbeschreibung als Ausgangspunkt definiert und als Softwarewerkzeug umgesetzt. Mit Hilfe dieser Funktionsbeschreibung kann sowohl die Struktur als auch das Verhalten von Maschinen und Anlagen spezifiziert werden. Dabei werden Beschreibungsmittel eingesetzt, die von den Entwicklern aller Fachdisziplinen genutzt werden können. Die Funktionsbeschreibung nimmt die Rolle eines mechatronischen Informationsmodells ein, in welches die für die VIBN relevanten Daten aus den CAD-Systemen rückgeführt werden. Im Zusammenwirken der Funktionsbeschreibung mit den CAD-Daten kann anschließend das Modell für die VIBN automatisiert abgeleitet werden.

**Towards an Integrated System Model for Testing and Verification [HB08]** Models and documents created during the development process of automation machines typically can be categorized into mechanics, electronics and software. The functionality of an automation machine is, however, realized by the interaction of all three of these domains. Thus, no single model covering only one development category will be able to describe the behavior of the machine thoroughly. For the early planning of machine design, virtual prototypes and especially for the formal verification of requirements, an integrated functional model of the machine is required. This paper introduces a technique, which can be used to model automation machines on an abstract level, including coarse-grained descriptions of mechanics, electronics and software aspects with special focus on modeling domain-specific issues, such as material flow and collision response. The resulting models are detailed enough to be simulated or verified but still suitably abstract to allow a fast creation and an efficient simulation.

**Interdisziplinäre Funktionsmodellierung im Anlagenbau [BHL09]** Moderne mechatronische Produktionssysteme sind durch ein Zusammenwirken der Disziplinen Mechanik, Elektrotechnik und Software gekennzeichnet. Vor allem in der Entwicklungsphase stellt die Integration der unterschiedlichen Fachbereiche eine Herausforderung für die Maschinenhersteller dar. Sequenziell aufgebaute Entwicklungsprozesse führen zu langen Entwicklungszeiten. Zusätzlich erschweren die mangelnde Dokumentation und Kommunikation die interdisziplinäre Zusammenarbeit zwischen den Fachabteilungen. Im Rahmen des Forschungsvorhabens AutoVIBN wurde diese Problemstellung aufgegriffen und eine Beschreibungstechnik entwickelt, die eine disziplinenunabhängige, funktionale Modellierung einer Anlage erlaubt. Ergänzende Softwarewerkzeuge unterstützen die Erstellung und frühzeitige Simulation dieser Modelle sowie die Nutzung der gesammelten Informationen für die Anlagenentwicklung.

**A Semantic Model for Computer-Based Spatio-Temporal Systems [Hum09]** Over the last decades a large number of techniques have been developed for modeling, analyzing, and verifying software. For embedded or computer-based systems, however, the software is only one part of the entire system, which often can not be regarded isolated. This makes it hard, if not impossible, to judge the correctness of software without a thorough understanding of its environment. A natural solution of this problem is not only capturing the software part but also the entire system by suitable design models. In order to be useful, such a model has to be supported by semantics, which unambiguously define its meaning. In this paper a semantic model is presented, which captures temporal and spatial aspects of a system. These are important if the system manipulates rigid objects, as typically found in the domain of industrial automation.

**Specifying the Worst Case - Orthogonal Modelling of Hardware Errors [BH09]** During testing, the execution of valid cases is only one part of the task. Checking the behavior in boundary situations and in the presence of errors is an equally important subject. This is especially true in embedded systems, where parts of a system's function are realized by sensors and actuators, which are subject to wear and defects. As testing with the real hardware is costly and hardware defects are hard to stimulate, such tests are often performed using behavior models of the system which allow to execute the controller software against simulated hardware and environment. However, these models seldom contain possible hardware errors, as this makes the models more complex and, thus, harder to create and maintain. This paper presents a modeling technique for the description of system errors without modifying the original model. Error specifications for individual system components are modeled separately and can be used to augment the system model.

**Simulationsmodelle für die virtuelle Inbetriebnahme [JBLZ09]** Der steigende Anteil an Steuerungssoftware im Maschinen- und Anlagenbau stellt Hersteller und auch Endanwender vor neue Herausforderungen. Mit Hilfe virtueller Maschinenmodelle kann die Qualität der Software vor der Inbetriebnahme und dem Produktionsanlauf gesichert werden. Der Aufbau solcher Simulationsmodelle ist sehr zeitintensiv. Wissenschaftler der TU München

beschreiben eine Methode, die eine Teilautomatisierung des Prozesses zur Modellerstellung erlaubt.

**Integrated Behavior Models for Factory Automation Systems [BHLH09]** Despite the large number of models for different purposes of factory automation systems, most of these models consider only particular and in many cases static aspects of these systems, for example, the geometry or the electrical components. There is a need for suitable description methods, which integrate these individual models into a superior behavior model, including the geometry and the handling of material. Furthermore, it is important that this model is linked to the more detailed domain specific models and is specified sufficiently formal, in order to allow an automated analysis. This paper provides a solution to this problem by introducing a model, which comprises both spatial structure and system behavior. In addition, it is based on a thorough mathematical theory. The approach is supported by a software tool, which allows an integration into the development process.

**Automatic Model Generation for Virtual Commissioning [ZL10]** Modern mechatronic production systems are characterized by the interaction of mechanical, electrical and software engineering. During the development process machine manufacturers are challenged by the integration of various disciplines. In this regard, the control software plays a vital role and is becoming increasingly important. Functionality is transferred to the software, as these solutions are more flexible and economical compared to mechanical systems. The interaction of the different engineering disciplines and its subsequent effects cannot be incorporated during the design phase, which results in errors appearing as late as during the commissioning. Due to the mentioned reasons testing of control software becomes more difficult. For this purpose, a software test during the early design phases is possible by using a virtual machine model. This approach is also known as virtual commissioning. Nevertheless, this kind of simulation is rarely used in industrial practice because of the extremely time-consuming modeling process. This article describes an approach how to automate the generation of simulation models for virtual commissioning. A functional model and CAD data are used in order to generate machine models automatically, which are expressed in C++.

## 2.3 Personal- und Mitteleinsatz

Bei dem beantragten Forschungsvorhaben handelte es sich um ein interdisziplinäres Projekt, für dessen erfolgreiche Bearbeitung die Zusammenarbeit von Industrieunternehmen, dem Institut für Werkzeugmaschinen und Betriebswissenschaften (*iwb*) sowie dem Lehrstuhl IV: Software und Systems Engineering der Technischen Universität München erforderlich war. Für die Umsetzung der Ziele wurden insgesamt vier wissenschaftliche Mitarbeiter betraut. Die Interdisziplinarität des Forschungsthemas sorgte dafür, dass die Arbeitspakete größtenteils von beiden Forschungsstellen bearbeitet werden mussten, jedoch mit unterschiedlicher Schwerpunktsetzung. Der projektbegleitende Ausschuss überwachte durch

regelmäßige Treffen den Projektfortschritt. Des Weiteren wurde von dessen Mitgliedern Personal für die Analysephase aus AP1 und zur Unterstützung der Fallstudien zur Verfügung gestellt.

Die finanziellen Mittel wurden gemäß dem Projektantrag eingesetzt, wobei keine Großgeräte angeschafft wurden.

# 3 Detaillierte Ergebnisse

## 3.1 AP1: Anforderungsanalyse bei den Projektpartnern

Das Ziel dieses Arbeitspakets war, die Partnerunternehmen, deren Entwicklungsprozesse und Probleme zu analysieren sowie die im Projekt eingesetzte Werkzeuglandschaft und das begleitende Fallbeispiel festzulegen.

In den folgenden Abschnitten werden die vorgefundenen Konstellationen kurz zusammengefasst und das Fallbeispiel grob skizziert. Abschließend wird aufgezeigt, welche Probleme in der Diskussion mit den Projektpartnern identifiziert wurden und welchen Beitrag das Projekt AutoVIBN zu einer möglichen Lösung leisten kann.

### 3.1.1 Entwicklungsprozesse

Aufgrund der Vielfältigkeit der Produktspektren finden bei allen analysierten Unternehmen kaum Entwicklungen im Serienbereich statt. Damit geht ein hoher Aufwand für die Anfertigung von kundenspezifischen Anlagen einher. Gesamtanlagen werden aus Modulen zusammengesetzt, die in Bezug auf ihre Funktion, die verwendeten Bauteile, die Fertigung und die Software einzeln getestet und validiert werden. Im besonderem Maße ist die Automation der Maschine durch die Wünsche und Anforderungen des Kunden geprägt, da diese an die Gegebenheiten vor Ort anzupassen ist. Es hat sich gezeigt, dass vor allem bei den Automatisierungslösungen die Anpassungen zu Problemen und damit zum zeitlichen Verzug in den Entwicklungsprozessen führen können. Die Ursachen hierfür differieren je nach Anwendungsfall. So führt ein steigender Kommunikationsaufwand mit den Zulieferern von Halb- und Fertigzeugen dazu, dass firmeninterne Abläufe nicht ausreichend synchronisiert werden können. Uneinheitliche Beschreibungsmittel bedingen in Angebotsphasen, dass keine exakte Spezifikation der Maschinenanforderungen formuliert werden kann. Daraus resultieren Inkonsistenzen zwischen den verschiedenen Entwicklungssträngen (Mechanik, Elektrik und Software), die erst bei der Integration bzw. der Inbetriebnahme augenscheinlich werden.

Bei genauerer Betrachtung der Entwicklungsprozesse der beteiligten Firmen lässt sich feststellen, dass diese bei allen untersuchten Kooperationspartnern Ähnlichkeiten aufweisen. Das Vorgehen ist dabei von einem sequenziellen Ablauf geprägt (siehe Abbildung 3.1, oben), wobei im Bereich der Softwareentwicklung bereits teilweise die VIBN zum Einsatz kommt und damit eine partielle Parallelisierung der Prozesse angestrebt wird (idealisierter Prozess, siehe Abbildung 3.1, unten). Zu Beginn einer Entwicklung werden zunächst die Anforderungen an eine neue Maschine oder Anlage formuliert. Die Eingangsinformationen hierfür liefern entweder eigene Vertriebsmitarbeiter, Kundengespräche, Servicemitarbeiter oder Vorstudien. Dieser Phase schließt sich die mechanische Konstruktion an. Dabei werden Kom-

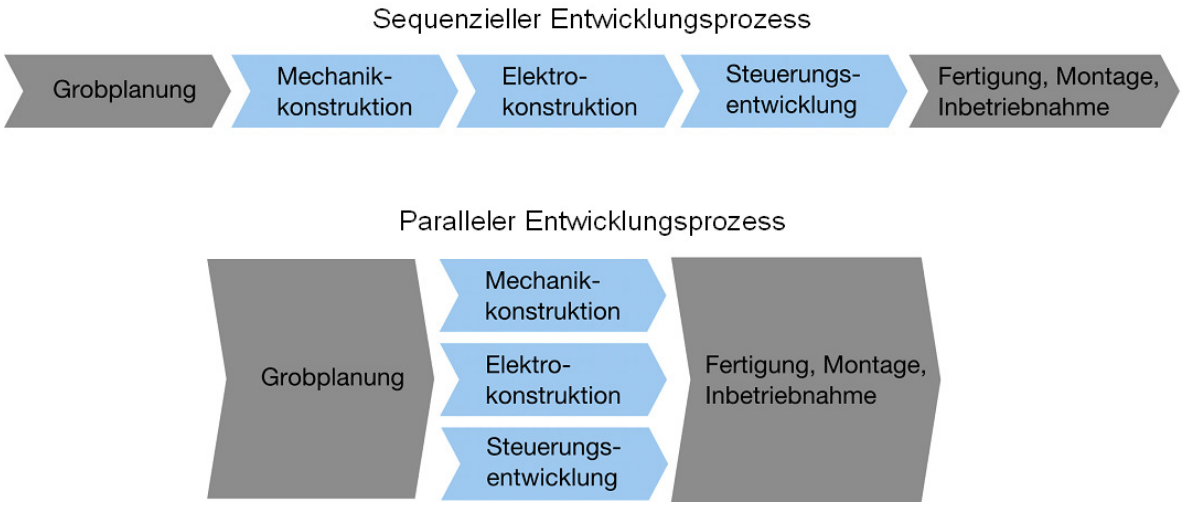



Abbildung 3.1: Sequenzieller und paralleler Entwicklungsprozess

ponenten festgelegt, mit welchen sich die angestrebten Funktionen realisieren lassen. Die elektrische Konstruktion greift die Ergebnisse auf und detailliert die Maschine weiter. Hierbei hat sich in den letzten Jahren der Trend abgezeichnet, aus vordefinierten Modulen fertige Schalt- und Installationspläne zu erzeugen. Diese werden lediglich mit zusätzlichen Konfigurationsdaten angereichert.

Im Anschluss daran beginnt die mit der Softwareentwicklung beauftragte Abteilung, die Funktionen des Steuerungssystems zu implementieren. Dabei wird meist direkt in der Entwicklungsumgebung des Steuerungsherstellers gearbeitet. Vereinzelt werden auch Ansätze verfolgt, herstellerunabhängige Plattformen zu nutzen, um Softwarefunktionen abzubilden. Diese müssen dann durch einen Compiler in das nötige Zielformat übersetzt werden. Wie bereits erläutert, setzten die untersuchten Hersteller vereinzelt die VIBN als Testmittel ein. Das Vorgehen ist dabei wie folgt ausgeprägt, zunächst werden die Modelle der Anlage aufgebaut und dann die Hardware- oder Software-in-the-Loop-Anordnung erstellt. Es findet eine starke Fokussierung auf den reinen Softwaretest statt, die Funktion an sich wird aufgrund des Projektfortschrittes nicht hinterfragt.

Typische Fehler die damit behoben werden können sind bspw.:

- Falsche oder fehlende Fehlermeldungen
- Fehlende Funktionsblöcke
- Falsche Adressierungen
- Fehler in der Benutzerführung des Maschinenprogramms

Durch einen früheren Einsatz der Simulationstechnik könnten auch funktionale Fehler aufwandsärmer behoben werden. Das würde auch konstruktive Änderungen vereinfachen, da sich diese zu Entwicklungsbeginn noch nicht derart kostenintensiv auswirken würden.

Generell lässt sich feststellen, dass die Entwicklungsprozesse nach wie vor stark von den Erfahrungen und dem Wissen einzelner Mitarbeiter abhängig sind. Damit können zum Teil

die durch Kommunikationsprobleme entstandenen Fehler an den Schnittstellen kompensiert werden. Jedoch sehen die Projektpartner an dieser Stelle großes Potenzial, die Fachabteilungen durch geeignete Methoden und Werkzeuge zu unterstützen. Übergreifende, interdisziplinäre Modelle könnten dazu einen Beitrag leisten.

### 3.1.2 Werkzeuglandschaft

Hinsichtlich der Werkzeuglandschaft soll an dieser Stelle zwischen den Konstruktions- und den Simulationswerkzeugen unterschieden werden. Zunächst wurde zusammengestellt, welche Systeme sich bei den Mitgliedern des projektbegleitenden Ausschusses im Einsatz befinden.

Werkzeuge im Konstruktionsbereich:

- OneSpace Designer, Fa. CoCreat (MCAD)
- Solid Edge, Fa. Siemens (MCAD)
- Pro/Engineer, Fa. PTC (MCAD)
- Solid Works, Fa. Dassault Systemes (MCAD)
- NX, Fa. Siemens (MCAD)
- $E^3$, Fa. Zuken (ECAD)
- Elcad, Fa. Autotec (ECAD)
- Eplan, Fa. Eplan (ECAD)

Die verwendeten Simulationswerkzeuge sind:

- WinMOD, Fa. Mewes und Partner
- Virtuos, Fa. ISG
- Ansys, Fa. Ansys
- VNCK, Fa. Siemens
- Sinumerik Machine Simulator, Fa. Siemens

Im Projektvorhaben AutoVIBN kommen als MCAD-Werkzeug von der Fa. Siemens das Produkt NX und als ECAD-Werkzeug das System $E^3$ von der Fa. Zuken zum Einsatz. Beide zeichnen sich durch umfangreiche Programmierschnittstellen aus, welche einen Zugriff auf die hinterlegten Modelle ermöglichen.

### 3.1.3 Fallbeispiel

Die im Rahmen des beschriebenen Forschungsprojektes erzielten Ergebnisse sollen mit Hilfe eines Anwendungsbeispiels verifiziert werden. Hierzu hat sich der projektbegleitende Ausschuss zunächst darauf geeinigt, ein fiktives Automatisierungssystem zu entwickeln, welches eine möglichst große Schnittmenge an Maschinenfunktionalität der am Projekt beteiligten Firmen darstellt. Dabei wird bewusst auf das Heranziehen einer bereits existierenden

Anlage verzichtet und eine komplette Neukonstruktion angestoßen, da hiermit die Integration der entwickelten Methode in den Entwicklungsprozess besser nachvollzogen werden kann. Das hier vorgestellte Beispiel wurde während der ersten Projektphasen eingesetzt. Etwa nach der Hälfte der Projektlaufzeit wurde vereinbart, das Vorgehen an zwei realen Beispielen aus der Praxis zu validieren. Diese sind in AP9 näher erläutert.

Im folgenden Teil werden die fiktiven Fallbeispiele grob skizziert. Es handelt sich dabei um erste Konstruktionsmodelle im CAD, die zur Diskussion mit dem projektbegleitenden Ausschuss genutzt wurden. Auf dieser Basis wurden dann die Anlagen im CAD weiter detailliert (siehe auch AP9).

### Funktionalität und konstruktive Anforderungen

Die gewünschte Funktionalität und die konstruktiven Anforderungen des Anwendungsbeispiels sind Ergebnisse des ersten Projekttreffens und der nachfolgenden Analysephase, bei der die Entwicklungsprozesse der Projektteilnehmer untersucht wurden. Nachfolgend sind diese aufgeführt.

Funktionen:

- Materialzu- und -abführung
- Material transportieren
- Material heben, drehen
- Material identifizieren
- Material vereinzeln
- Positionieren
- Position erfassen

Daraus ergeben sich folgende Anforderungen:

- Längs- und Quertransport durch Förderbänder
- Kinematiken für Positionieraufgaben (NC-Achsen)
- Positionserfassung durch Absolutwert- und Drehgeber
- Schrittkettenfunktionen für die Materialhandhabung
- Berücksichtigung der Zustandsänderung des Materials
- Identifikation des Materials
- Endlagenschalter, Lichtschranken
- Zylindern, Stopper

Da vor allem die Änderungen im laufenden Entwicklungsprozess für die hier entwickelte Methode von Interesse waren, wurde das Anwendungsbeispiel kontinuierlich erweitert. Hierzu wurden vier Varianten des Transportsystems konstruiert, die aufeinander aufbauen und im Folgenden vorgestellt werden.

### Variante 1: Förderbandsystem

Die erste Variante ist ein einfaches Transportsystem, welches aus zwei aneinandergereihten Förderbändern besteht, siehe Abbildung 3.2.

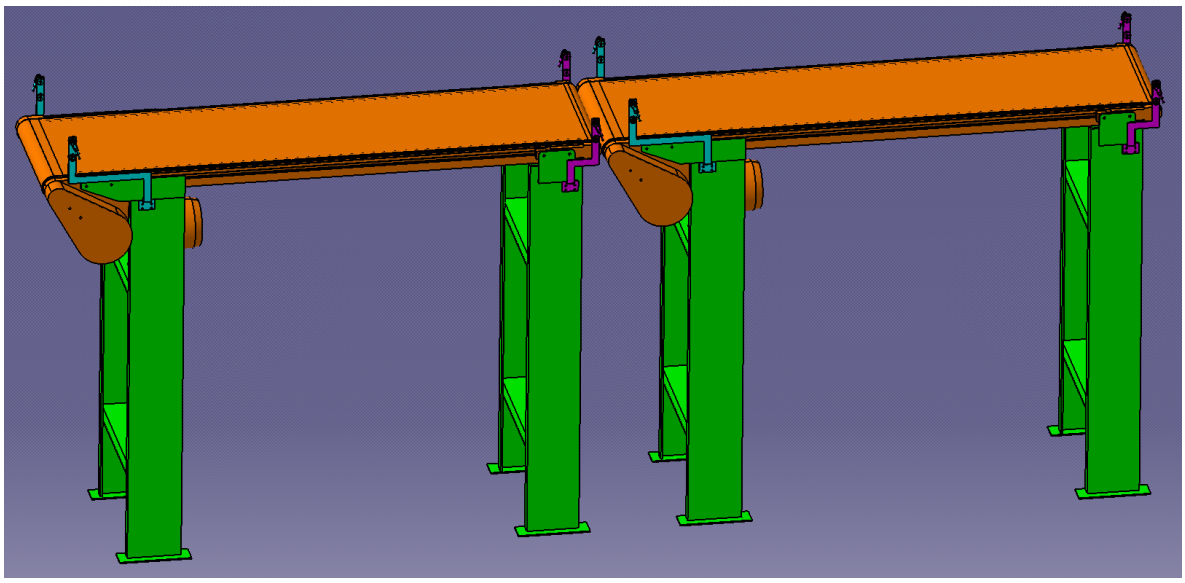


Abbildung 3.2: Variante 1 der Beispielanlage

### Variante 2: Erweitertes Förderbandsystem

Die zweite Variante basiert auf Variante 1 und wurde um ein zusätzliches Förderband ergänzt, siehe Abbildung 3.3. Durch die stetige Erweiterung der Beispielanlage konnte in den nachfolgenden Arbeitspaketen untersucht werden, wie sich Änderungen, z. B. durch Kundenanforderungen, im laufenden Entwicklungsprozess auswirken.

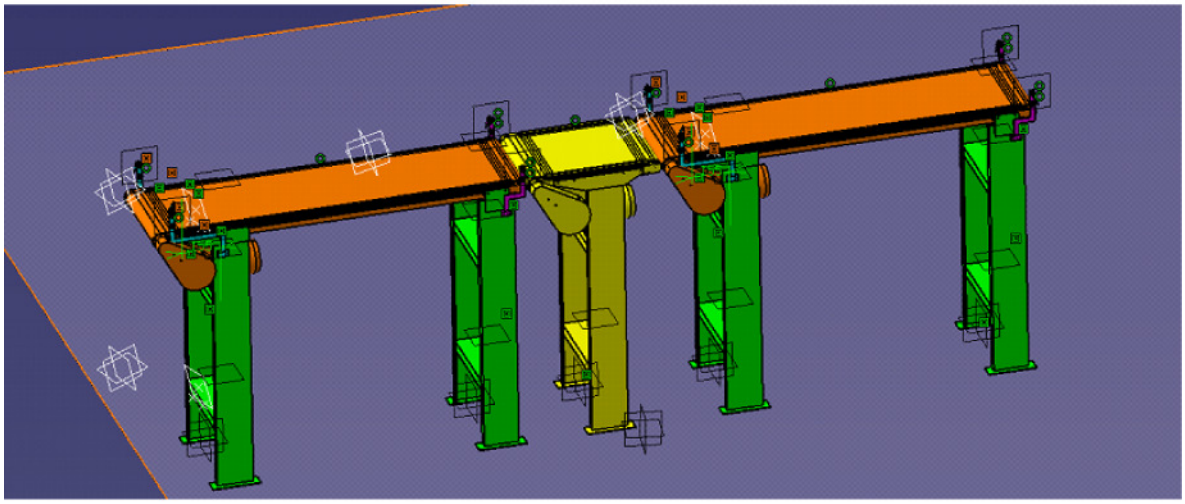


Abbildung 3.3: Variante 2 der Beispielanlage

**Variante 3: Förderbandsystem mit Hub-/Drehstation**

Die Beispielanlage wurde in einem nächsten Schritt um eine Hub-/Drehstation erweitert, siehe Abbildung 3.4.

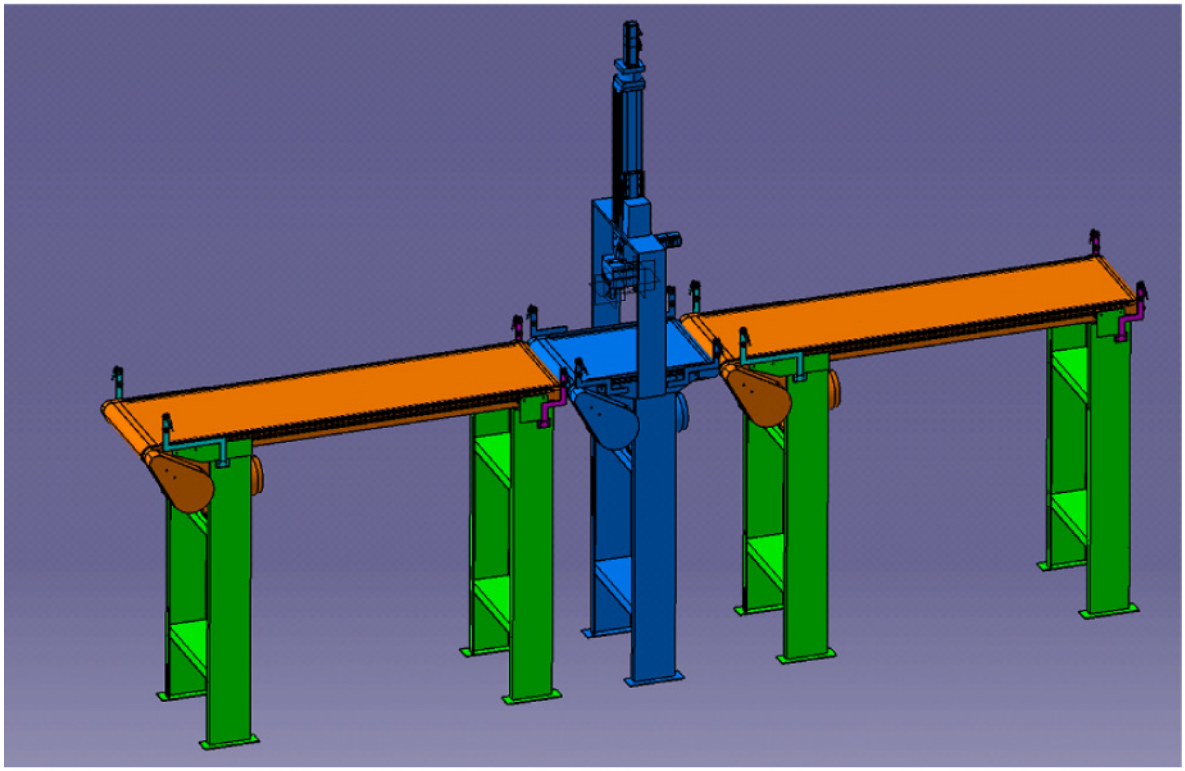


Abbildung 3.4: Variante 3 der Beispielanlage

#### Variante 4: Förderbandsystem mit Portal

In der letzten Ausbaustufe wurde die Hub-/Drehstation durch ein Portal ersetzt, mit welchem Teile vom Band abgegriffen und zu einer Messstation transportiert werden konnten, siehe Abbildung 3.5. Auf dem dritten Band wurden zusätzlich ein Stopper und ein Abschiebemechanismus für Ausschussteile integriert.

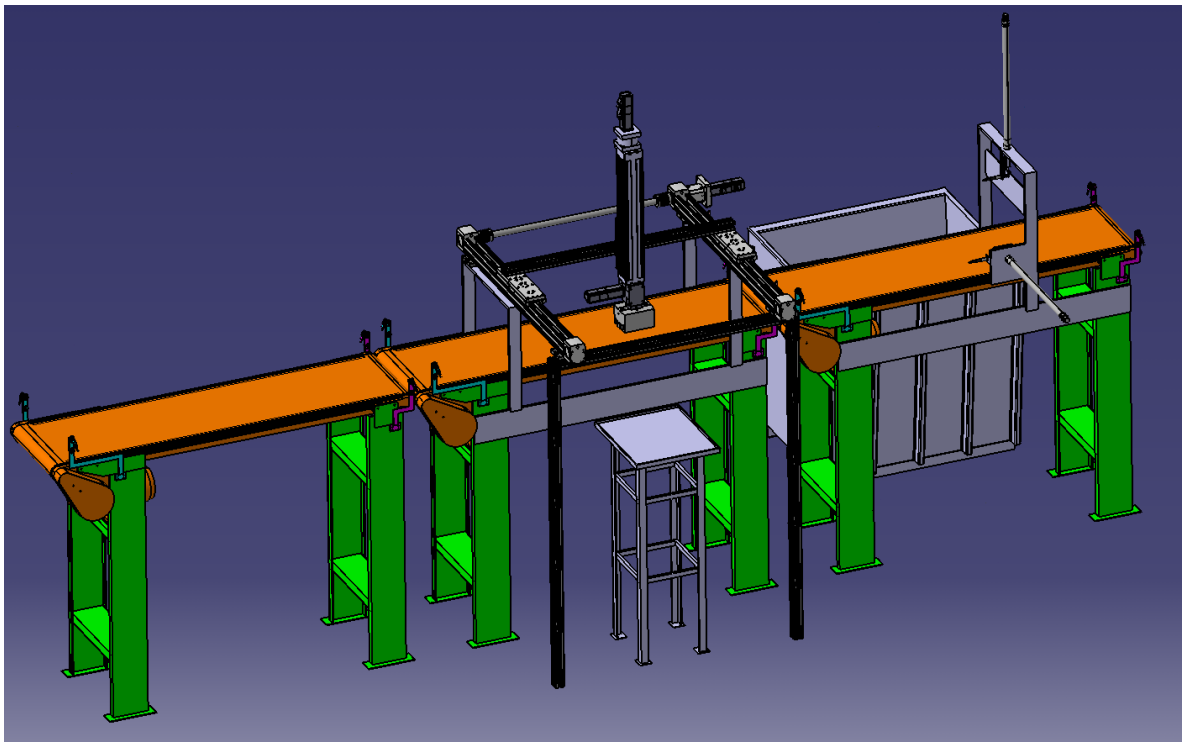


Abbildung 3.5: Variante 4 der Beispielanlage

### 3.1.4 Identifizierter Projektbeitrag

In diesem Abschnitt werden die Kernprobleme beschrieben, die zusammen mit den Projektpartnern diskutiert wurden und für die das Projekt AutoVIBN einen Lösungsbeitrag leisten könnte.

#### Verbesserte Abstimmung zwischen den Disziplinen

Die Kommunikation zwischen den verschiedenen an der Entwicklung einer Maschine beteiligten Disziplinen, wie bspw. der Mechanik, der Elektrik und der Informatik, gestaltet sich aufgrund der gegebenen Strukturen noch immer problematisch. Die gemeinsame Schnittstelle wird, abhängig vom Unternehmen, mit verschiedenen Mitteln, wie z. B. strukturiertem oder unstrukturiertem Text, Excel-Tabellen, Eingabe- und Ausgabelisten oder CAD-Modellen, spezifiziert. Diesen Beschreibungsmitteln ist gemein, dass sie entweder unpräzise

sind oder nur die Sichtweise einer einzelnen Disziplin enthalten. In der Konsequenz werden viele potenzielle oder tatsächliche Probleme oft nur zufällig oder erst bei der Zusammenführung der Teilergebnisse während der Inbetriebnahme gefunden, was verglichen mit einem frühen Erkennen des Fehlers zu höheren Änderungskosten führt. Diese Herausforderungen sind zum Teil nur dadurch beherrschbar, dass die entwickelten Maschinen Ähnlichkeiten aufweisen und die fehlenden Schnittstelleninformationen durch die Erfahrung der Mitarbeiter interpoliert werden.

Eine mögliche Verbesserung könnte durch einen frühzeitigen Einsatz einer formalen Modellierungstechnik erzielt werden. Die Verwendung der Beschreibungsmittel aus der bestehenden FWF-Richtlinie zur Funktionsbeschreibung [FWF03] scheiterte bisher vor allem daran, dass keine geeigneten Werkzeuge zur Verfügung standen und für die verschiedenen Fachabteilungen der Nutzen einer zusätzlichen Dokumentation nicht ersichtlich war.

Die weiterentwickelte Funktionsbeschreibung, wie sie im Arbeitspaket 2 definiert wird, wäre für diese Zwecke und insbesondere für die Unterstützung der Kommunikation während der Entwicklung einer Maschine geeignet. Durch den im Forschungsvorhaben AutoVIBN entwickelten Editor steht auch eine entsprechende Werkzeugunterstützung bereit. Es handelt sich jedoch lediglich um ein Funktionsmuster, kein serienreifes Produkt. Der Aufwand für die Erstellung eines funktionalen Maschinenmodells lässt sich einerseits durch die höhere Qualität des Entwicklungsergebnisses und die Einsparungen durch das frühere Identifizieren von Fehlern rechtfertigen. Andererseits wurden im Laufe des Projektes Methoden erarbeitet, die mit Hilfe dieses funktionalen Modells die Konsistenz zwischen verschiedenen Entwicklungsartefakten sicherstellen und zu einer vereinfachten Erstellung von Simulationsmodellen führen.

#### Kostenreduzierte Erstellung von Simulationsmodellen

Alle Projektpartner, die bereits Erfahrungen mit der VIBN sammeln konnten, kritisieren, dass die Erstellung der Simulationsmodelle sich sehr zeitintensiv gestaltet. Dies führt dazu, dass die Modelle zu spät im Entwicklungsprozess zur Verfügung stehen und dadurch ein reeller Mehrwert verhindert wird. Zudem sind die Kosten für die Modellerstellung zu hoch. Des Weiteren müssen viele Informationen redundant eingegeben werden, die bereits in anderen Modellen vorliegen.

Diese Problemstellungen zu lösen, war das Hauptziel des Forschungsvorhabens AutoVIBN. Durch die Einbeziehung existierender Modelle sollte die Modellerstellung beschleunigt werden. Zudem ließen sich durch die Kombination verschiedener Sichten (Modelle) auf die Maschine, zusammen mit einer Funktionsbeschreibung als integrierendes Modell, zusätzliche Verhaltensinformationen in das Simulationsmodell übertragen.

#### Frühzeitige Validierung der Entwicklungsergebnisse

Je früher die Steuerungssoftware im Entwicklungsprozess getestet werden kann, desto eher lassen sich Fehler zeitnah finden und beheben. Des Weiteren können mögliche Missverständnisse bei der Schnittstellenbeschreibung rechtzeitig eliminiert und dadurch aufwändige

Nacharbeiten bzw. Iterationsschleifen minimiert werden. Mit einem reduzierten Modellerstellungsaufwand für die VIBN lassen sich die Simulationsmodelle frühzeitig ausleiten, was den Steuerungsentwicklern ein umfangreiches Testen ermöglicht.

#### Testen von destruktiven Fehlerszenarien

Bestimmte Fehlerszenarien können an der realen Maschine nicht (bzw. unter hohem Risiko) oder nur mit erheblichem Aufwand getestet werden. Als Beispiele hierfür seien das Reißen von Antriebsriemen oder Kollisionen nach dem Ausfall von Sensoren genannt. Diese Fälle können mit Hilfe einer VIBN gefahrlos und aufwandsarm geprüft werden, was jedoch an sich keinen besonderen Beitrag dieses Projektes darstellt. Allerdings sollen die Anpassung des Modells für verschiedene Fehlerszenarien (z. B. der Ausfall eines Sensors) und das systematische Ableiten von Testfällen aus der Funktionsbeschreibung durch die Ergebnisse des Vorhabens unterstützt werden.

### 3.1.5 Fazit

Ihm Rahmen von AP1 wurden die Entwicklungsprozesse der am Projekt beteiligten Unternehmen sowie deren Werkzeuglandschaft analysiert. Daraus wurde ein Vorschlag unterbreitet, wie sich die entwickelte Methode in den Entwicklungsprozess der Unternehmen einbetten lässt, siehe AP9. Des Weiteren wurde ein Fallbeispiel zur Erprobung festgelegt. Insgesamt sollte durch die frühe Befragung und Einbindung der Projektpartner vermieden werden, dass die Ergebnisse des Forschungsprojektes an den Erfordernissen der Praxis vorbei zielen.

## 3.2 AP2: Funktionsbeschreibung für die Maschinenmodellierung

In diesem Arbeitspaket wurde eine Funktionsbeschreibung für mechatronische Systeme im Anlagen- und Werkzeugmaschinenbau entwickelt, mit der sich das integrierte Verhaltensmodell darstellen lässt. Hierbei wurde besonderer Wert darauf gelegt, dass die Semantik der Modellelemente und ihr Zusammenwirken klar definiert sind. Zudem wurde eine geeignete, textuelle und grafische Syntax erarbeitet, die ein Aufschreiben der Modelle ermöglicht, aber auch in einem Modellierungswerkzeug umgesetzt werden kann. Die in diesem Teil des Dokuments vorgestellte Modellierung ist nicht ausschließlich im Rahmen von AP2 entstanden, sondern vielmehr über den gesamten Projektverlauf fortwährend erweitert und verbessert worden. Um das Verstehen zu vereinfachen, wird hier jedoch der vollständige Modellumfang vorgestellt, was den im Vergleich zu den anderen Abschnitten höheren Umfang erklärt.

Im folgenden werden die Bestandteile des Meta-Modells eingeführt. Die zugrundeliegende mathematische Theorie, die sich auf den strombasierten Formalismus FOCUS [BS01] abstützt, wird in [Hum09] erklärt und daher hier nicht ausgeführt. Das Meta-Modell lehnt sich an das von AutoFOCUS [BHS99, SPHP02] an, welches eine Werkzeugumsetzung der FOCUS-Theorie ist.

Falls erforderlich werden die beschriebenen Teile des Meta-Modells durch Klassendiagramme aus der UML [BRJ98, RJB98] ergänzt. Diese enthalten nicht alle Details, um die Übersichtlichkeit zu waren. Fehlende Informationen, insbesondere Konsistenzbedingungen, sind im Text erklärt, da eine Formalisierung (etwa mittels OCL) häufig schwerer zu durchdringen ist. Das Modell wird teilweise auch durch Abbildungen ergänzt, die der experimentellen Werkzeugumsetzung entstammen. Die Bilder dienen als Beispiele für die verwendete, grafische Syntax.

### 3.2.1 Typen und Typsystem

Das Thema Typen und Typsysteme wurde in der Informatik in den vergangenen Jahrzehnten sehr ausführlich behandelt. Da der Schwerpunkt der Funktionsbeschreibung in der Modellierung des Verhaltens liegt, sind die genauen Details des verwendeten Typsystems von untergeordnetem Interesse. Daher wurde ein sehr einfaches und klar definiertes Typsystem gewählt, das dennoch ausdrucksstark genug ist, um Aufzählungstypen, Record-Typen (auch als Tupel-Typen bekannt) und rekursive Typen (wie etwa für Listen und baumartige Strukturen benötigt) abzubilden. Das verwendete Typsystem basiert auf dem des Werkzeugs AutoFOCUS3, das aktuell am Lehrstuhl für Software & Systems Engineering entwickelt wird.

#### Typen

Ein Typ wird über einen Namen und eine Menge von Konstruktoren spezifiziert. Die Konstruktoren bestehen jeweils wiederum aus einem Namen und einer Liste von Typen, die als Parameter bezeichnet werden. Die Trägermenge eines Typs definiert sich induktiv über seine Konstruktoren, wobei gültige Elemente der Parametertypen eingesetzt werden.

Ein Konstruktor wird derart aufgeschrieben, dass die geklammerte Parameterliste seinem Namen folgt. Bei einem Konstruktor ohne Parameter können die (dann leeren) Klammern auch entfallen. Für einen Typen schreibt man erst seinen Namen, gefolgt vom Gleichheitszeichen und der Liste der Konstruktoren, getrennt durch vertikale Striche |. Die Liste wird durch ein Semikolon abgeschlossen. Üblicherweise beginnen Typnamen mit einem Großbuchstaben, während die Namen von Konstruktoren mit einem kleinen Buchstaben anfangen. Die Definition von Typen wird im folgenden Beispiel verdeutlicht.

**Beispiel 1.** *Die einfachsten Typen sind solche, die nur leere Konstruktoren besitzen, sog. Aufzählungstypen.*

```
Color = red | green | blue ;
Direction = north | east | south | west ;
```

*Die Trägermenge (also die Menge der gültigen Elemente) von* color *ist als* $\{red, green, blue\}$ *definiert, die von* direction *als* $\{north, east, south, east\}$*. Typen, die durch rekursionsfreie Konstruktoren beschrieben werden, d. h. kein Konstruktor verweist auf den Typen selbst oder einen anderen Typen der indirekt wieder auf diesen Typen verweist, werden meist Record- oder Tupel-Typen genannt:*

```
DirectionTriple   = dirTriple (Direction, Direction, Direction) ;
ComplexDirection = noDirection | simpleDirection (Direction) |
                   colorDirection (DirectionTriple, Color) ;
```

*Die Trägermenge von* DirectionTriple *enthält die Elemente dirTriple (North, North, West) und dirTriple (South, South, South). Die Trägermenge von* ComplexDirection *umfasst noDirection, simpleDirection(East) und colorDirection ( dirTriple (North, North, West), red). Typen, wie bspw.* ComplexDirection*, die mehrere recordartige Konstruktoren vereinen, werden manchmal auch Variantenttypen oder Vereinigungstypen genannt. Hebt man zudem die Beschränkung der Rekursion auf, so lassen sich mit dem Typsystem auch Listen und baumartige Strukturen beschreiben.*

```
ColorList = emptyColorList | colList (Color, ColorList) ;
Tree      = emptyTree | consTree (Tree, ColorList, Tree) ;
```

*Während die vorherigen Beispiele alle endliche Trägermengen hatten, sind die Trägermengen von* ColorList *und* Tree *unendlich. Der Typ* ColorList *enthält demnach z. B. emptyColorList, colList (red, emptyColorList) und colList (blue, colList (green, emptyColorList)).*

Die primitiven Typen *boolean* (true und false), *int* (Ganzzahlen), und *double* (Fließkommazahlen) werden als gegeben vorausgesetzt. Zudem wird für jeden definierten Typen ein (struktureller) Äquivalenztest mittels $==$ angenommen.

### Funktionen

Die Typdefinition ist ausreichend, um die Trägermenge zu charakterisieren, allerdings sollen mit diesen Typen auch Berechnungen durchgeführt werden können. Hierfür werden Funktionen definiert, die Regeln beschreiben, wie eine Liste von Termen in einen neuen Term umgewandelt wird. Eine Funktion wird über ihren Namen, eine Parameterliste (wie bei den

Konstruktoren) und eine Liste von Transformationsregeln beschrieben. Jede Transformationsregel ist durch eine Liste von Mustertermen und einen Ergebnisterm definiert. Die Musterterme dürfen nur aus Konstruktoren und Variablen bestehen, während der Ergebnisterm auch weitere Funktionsaufrufe enthalten kann. Die Eingabe einer Funktion basiert auf einer Liste von Elementen aus den Trägermengen der entsprechenden Typen in der Parameterliste. Eine Transformationsregel heißt *passend* für eine Eingabe, wenn eine Belegung der freien Variablen in den Mustertermen existiert, so dass dieser dem entsprechenden Eingabeterm gleicht. Das Funktionsergebnis wird durch den Ergebnisterm der ersten passenden Transformationsregel definiert, wobei die enthaltenen Variablen entsprechend der verwendeten Belegung substituiert werden.

Diese Vereinbarungen führen dazu, dass das Funktionsergebnis immer eindeutig bestimmt ist, solange mindestens eine passende Transformationsregel existiert. Im Kontext der Funktionsbeschreibung werden nur *totale* Funktionen erlaubt, d. h. solche, die für jede syntaktisch gültige Eingabe eine passende Transformationsregel besitzt, deren Ergebnisterm nach endlich vielen Schritten berechnet ist, also terminiert.

Hinsichtlich der Notation werden Funktionen durch ihren Namen, gefolgt von der geklammerten Parameterliste, einem Doppelpunkt und dem Ergebnistyp deklariert. Die Transformationsregeln folgen danach und werden durch vertikale Linien | voneinander und von der Funktionsdeklaration getrennt. Die Musterterme und der Ergebnisterm werden durch –> separiert und die gesamte Funktionsdefinition wird durch ein Semikolon abgeschlossen. Die Namen von Variablen beginnen zur besseren Unterscheidung mit einem Unterstrich, während Funktionsnamen mit einem kleinen Buchstaben anfangen.

**Beispiel 2.** *Im Folgenden wird auf die Typen aus Beispiel 1 aufgebaut. Eine Funktion, die die nächste Richtung im Uhrzeigersinn angibt, lässt sich wie folgt beschreiben:*

```
clockwise (Direction) : Direction
  | north –> east
  | east  –> south
  | south –> west
  | west  –> north ;

opposite (Direction) : Direction
  | _d    –> clockwise (clockwise (_d)) ;
```

*Die zweite Funktion wendet die Funktion* clockwise *zweimal an. Ein Beispiel mit mehreren Mustertermen ist die if-then-else-Funktion, die ihren zweiten oder dritten Parameter zurückliefert, abhängig vom Wert des ersten booleschen Parameters.*

```
ite (boolean, ColorList, ColorList) : ColorList
  | true,  _a, _b  –>  _a
  | false, _a, _b  –>  _b  ;
```

*Mittels rekursiver Funktionen lassen sich auch komplexere Aufgabenstellungen lösen, die im Zusammenhang mit rekursiven Typen auftreten. Die folgende Funktion entfernt das erste Auftreten einer gegebenen* color *aus einer Liste.*

```
delColor (ColorList, Color) : ColorList
  | emptyColorList, _c      –>  emptyColorList
  | colList (_c1, _l), _c2  –>
```

```
ite (_c1 == _c2, _l, colList (_c1, delColor (_l, _c2))) ;
```

Für die primitiven Typen werden die üblichen Operationen (boolesche Verknüpfungen, Vergleiche und Grundrechenarten) als gegeben vorausgesetzt. Für diese Operationen wird die aus der Programmiersprache C bekannte Infix-Notation und Auswertungsreihenfolge verwendet.

### 3.2.2 Räumliche Informationen

Das räumliche Modell basiert auf dem dreidimensionalen euklidischen Raum. Objekte in diesem Raum werden durch eine Menge einzelner (transformierter) geometrischer Primitive (etwa Quader, Zylinder und Kugeln) beschrieben. Zur Tranformation werden alle formerhaltenden Operationen zugelassen, also beliebige Kombinationen aus Translationen und Rotationen. Prinzipiell lässt sich das räumliche Modell so erweitern, dass auch komplexere Geometrien unterstützt werden. Dies wird an dieser Stelle jedoch wegen der Übersichtlichkeit unterlassen. Es sei angemerkt, dass im Kontext der Funktionsbeschreibung eine *Kollision* immer eine echte (volldimensionale) Durchdringung der geometrischen Primitive und nicht eine bloße Berührung meint.

Das eigentliche Meta-Modell ist in Abbildung 3.6 zu sehen und fasst den zuvor beschriebenen Sachverhalt zusammen. Durch das Element *ShapeGroup* ist eine hierarchische Strukturierung der geometrischen Objekte möglich.

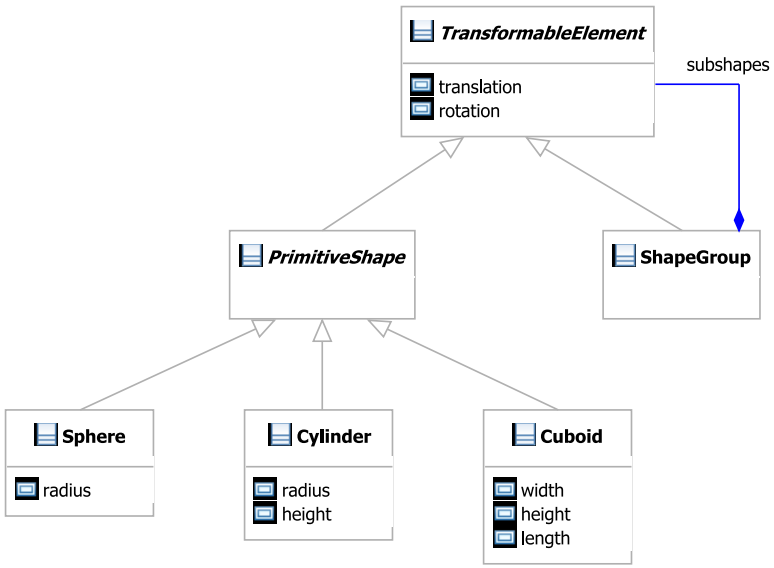



Abbildung 3.6: Meta-Modell zur Erfassung räumlicher Informationen

### 3.2.3 Statische Aspekte

Das Kernelement des Modells ist die *Komponente*. Diese wird verwendet, um einzelne Teile oder auch das vollständige System zu beschreiben. Die syntaktische Schnittstelle einer Komponente definiert ihre statischen Anteile. Deren Hauptelemente sind in Abbildung 3.7 zu sehen. *Ports* legen die Kommunikationsendpunkte der Komponenten fest, wobei Kommunikation sowohl den Signalaustausch über einen Feldbus als auch die „Übertragung“ des Drehmomentes eines Motors zum angetriebenen Band umfasst. Die Definition eines Ports enthält den Typ, der für ausgetauschte Nachrichten benutzt wird, und einen initialen Wert, der im ersten Simulationstakt anliegt. Die räumliche Ausprägung der Komponente ist durch die *Parts* definiert, die die geometrische Ausdehnung beschreiben. Diese Anteile des Systems werden außerhalb der Funktionsbeschreibung üblicherweise in einem MCAD-Werkzeug modelliert. Stellen im Raum, an denen eine Komponente die Anwesenheit von anderen Parts erkennen und entsprechend reagieren kann, werden durch *Detectors* markiert. Bereiche an denen mit Material interagiert wird, kennzeichnet ein *Binding*, das in Abschnitt 3.2.6 näher erklärt wird. Das Element *Mover* beschreibt Möglichkeiten, um auf die Position anderer Teile oder des Materials einzuwirken.

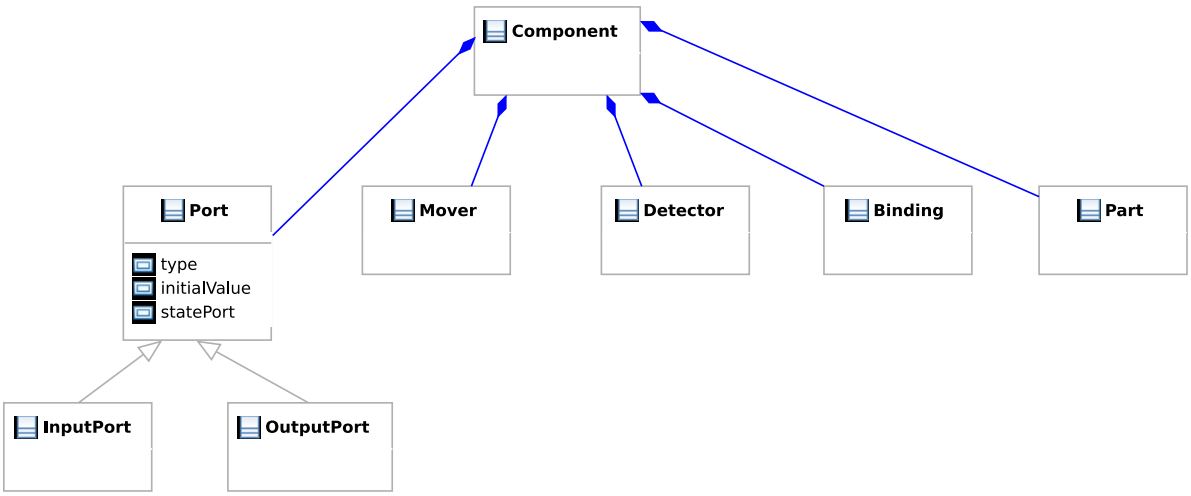



Abbildung 3.7: Meta-Modell zur syntaktischen Schnittstellendefinition von Komponenten

#### Signal- und Zustandsports

In reinen Softwaresystemen wird meist eine ereignisbasierte Kommunikation genutzt, d. h. Nachrichten werden nur als Reaktion auf Ereignisse oder Zustandsänderungen verschickt. Oft wird daher auch keine Nachricht gesendet. Das Gegenstück ist die zustandsbasierte Kommunikation, bei der der aktuelle Zustand periodisch ausgestrahlt wird. Dieses Verhalten findet sich häufig bei einfachen Sensoren wieder, es lässt sich aber auch zur Modellierung rein mechanischer Übertragungselemente nutzen.

Da sowohl die Hardware als auch die Software durch das Modell abzudecken sind, werden beide Arten der Kommunikation unterstützt. Ob eine ereignis- oder zustandsbasierte Ausprägung verwendet wird, muss am Port im booleschen Attribut *signalPort* hinterlegt werden. Solange nur Ports verbunden werden, die das gleiche Kommunikationsparadigma nutzen, bestimmt dieses sog. Flag nur, ob die *leere Nachricht* $\epsilon$, die die Abwesenheit von Nachrichten im Modell darstellt, verschickt werden kann. Es lassen sich aber auch Ports mit verschiedenen Kommunikationsparadigmen verknüpfen. Wenn der Sender zustandsbasiert ist, stellt dies kein Problem dar, da der Empfänger auch funktioniert, wenn keine $\epsilon$-Nachrichten vorkommen. Der umgekehrte Fall bedarf jedoch gesonderter Behandlung, da der Sender $\epsilon$ verschicken kann, was der Empfänger allerdings nicht versteht. Dieser Fall wird so abgedeckt, dass der Empfänger die letzte Nachricht, die nicht $\epsilon$ war, speichert und diese wiederverwendet, falls ein $\epsilon$ empfangen wird.

### Integration von Geometrie

Ein wesentliches Ziel der Funktionsbeschreibung ist die Integration von räumlichen Aspekten und Verhalten. Das Bindeglied stellen die Parts, Detectors und Bindings dar, die einer Komponente zugeordnet sind und eine geometrische Repräsentation besitzen. Abbildung 3.8 zeigt, wie die Modellelemente aus den Abbildungen 3.6 und 3.7 im Meta-Modell zusammengeführt werden. Durch das Erben von *TransformableElement* können einer Komponente lokale Transformationen zugewiesen werden. Die Parts, Detectors und Bindings werden als Container für geometrische Objekte gestaltet, indem sie von *ShapeGroup* erben.

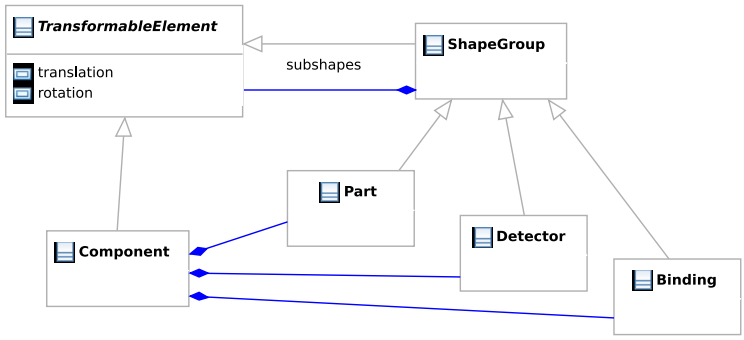



Abbildung 3.8: Meta-Modell zur Integration von Geometrie und Komponenten

Durch dieses Meta-Modell besitzen Parts, Detectors und Bindings eine feste Form, ihre Position (Transformation) kann sich aber über die Zeit ändern. Es gibt Fälle, bei denen auch die Form eines Teils variieren sollte. Ein Beispiel dafür ist die Deformation eines Parts nach einer Kollision in Folge einer Störsituation in der Anlage. Des Weiteren können Änderungen am Material durch Verarbeitungsschritte, wie z. B. Bohren oder Fräsen, auftreten. Eine mögliche Lösung besteht darin, für Parts, Detectors und Bindings mehrere Formen zu definieren und verhaltensbasiert zwischen diesen umzuschalten. Noch leistungsfähiger wäre

die Integration bestehender Lösungen aus der FE- oder Abtragssimulation in die Verhaltensbeschreibung. Diese Ansätze wurden im Forschungsprojekt nicht verfolgt, bieten aber Potenzial für zukünftige Arbeiten.

### Movers und Achsen

Die Transformationen, die ein Mover auf andere Elemente ausüben kann, werden mittels Achsen beschrieben. In Abbildung 3.9 wird diese Eins-zu-eins-Beziehung zwischen Mover und Achse verdeutlicht. Die Details, welche Modellteile von einem Mover und seiner Achse beeinflusst werden und wie stark dessen Position geändert wird, ist in Abschnitt 3.2.4 näher erläutert. An dieser Stelle liegt der Fokus auf den statischen Aspekten, die vor allem die möglichen Transformationen umfassen.

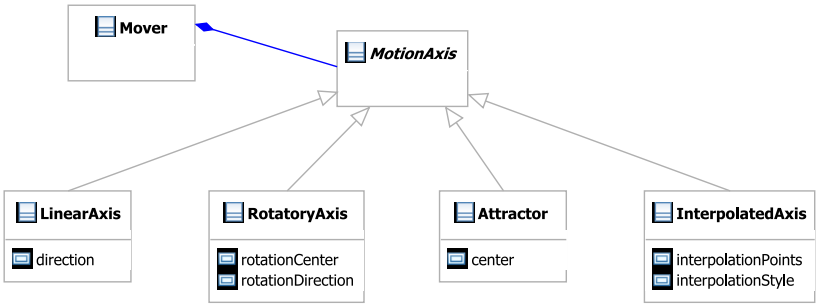



Abbildung 3.9: Meta-Modell für die Mover-Definition mit Hilfe von Achsen

Es existieren zwei triviale Arten der Transformation, die unabhängig vom betrachteten Element sind: lineare und rotatorische Bewegung. Diese werden durch die *LinearAxis* und die *RotatoryAxis* im Meta-Modell beschrieben. Theoretisch lässt sich jede Bewegung durch Kombinationen der beiden superpositionieren. Allerdings sind damit bestimmte Bewegungspfade nur sehr umständlich darstellbar. Beispiele hierfür sind die Materialbewegung auf einem mäanderförmigen Transportband oder Kettenmagazine. Um die Modellierung solcher Fälle zu vereinfachen, wird die Definition von Bewegungspfaden mit Hilfe von Stützpunkten ermöglicht (*InterpolatedAxis*), die dann nach einem wählbaren Schema interpoliert werden (z. B. B-Splines). Die auf ein Element angewandte Transformation hängt von der relativen Position zum Pfad ab. Die räumliche Manipulation wird so gewählt, dass eine Vorwärtsbewegung des Elements entlang des Pfades erzielt wird.

### Grafische Syntax

Die grafische Syntax, die für die Definition der syntaktischen Anteile der Komponente gewählt wurde, ist in Abbildung 3.10 beispielhaft zu sehen. Eine Komponente wird als Rechteck dargestellt, mit den Ports als kleinen Kreisen auf dessen Rand. Wenn nötig können die Ports entsprechend beschriftet werden, um sie zu unterscheiden. Weitere Informationen

sind in der Tabelle rechts enthalten, die die syntaktischen Elemente der Komponente zusammen mit deren Namen aufführt. Die geometrischen Daten der Parts, der Detectors und der Bindings sowie die genaue Lage der linearen Achse sind dort nicht ersichtlich. Da diese durch Koordinaten im dreidimensionalen Raum beschrieben werden, ist eine Bearbeitung und Betrachtung in einem entsprechenden Werkzeug vorzuziehen.


| **comp** StopperUnit | |
|---|---|
| **inport:** | mode: EMode |
| **inport:** | request: **boolean** |
| **outport:** | close_stopper: **boolean** |
| **outport:** | part_present: **boolean** |
| **part:** | stopper_part |
| **detector:** | _sense1 |
| **binding:** | _stop_binding |
| **mover:** | _move_stopper: linear axis |

Abbildung 3.10: Syntaktische Komponentenspezifikation

### 3.2.4 Dynamische Aspekte

Der Schwerpunkt des Modells liegt in der Verhaltensbeschreibung, weshalb die Erfassung der dynamischen, über die Zeit veränderbaren Teile zentral behandelt wird. Im vorliegenden Modell stellen die Automaten die übergreifende und flexibelste Technik zur Verhaltensbeschreibung dar. Da bisherige Erfahrungen gezeigt haben, dass bestimmte Probleme mit Hilfe der endlichen Automaten nicht bzw. nur umständlich abbildbar sind, wurde das Meta-Model so ausgelegt, dass sich auch alternative Verhaltensspezifikationstechniken integrieren lassen.

Das Grobmodell für die Verhaltensdefinition berücksichtigt dies, wie in Abbildung 3.11 ersichtlich. Eine Komponente wird durch eine *ComponentBehaviorSpec* spezifiziert, von der es wiederum mehrere Implementierungen gibt, wobei der Automat nur eine von mehreren Möglichkeiten ist. Zusätzlich unterstützt das Modell die Verhaltensmodifikation durch eine sog. *BehaviorExtension*. Die Kernidee dabei ist, dass eine (nicht zusammengesetzte) Komponente durch genau eine primäre Verhaltensspezifikation definiert wird, aber mehrere orthogonale Verhaltensmodifikationen besitzen kann. Die im Diagramm gezeigte Aggregation *subcomponents* und die Klasse *Channel* werden erst im Zusammenhang mit der Komposition in Abschnitt 3.2.5 erklärt.

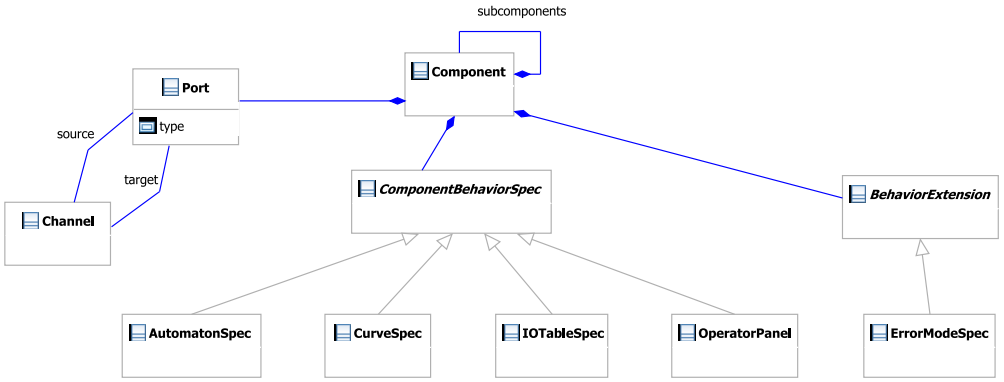



Abbildung 3.11: Meta-Modell für die Definition der semantischen Schnittstelle der Komponente

Die folgenden Absätze erläutern die Automaten detaillierter und geben einen kurzen Exkurs in die alternativen Spezifikationstechniken. Auch die Verhaltensmodifikation wird kompakt dargestellt. Zuvor soll jedoch die Integration von Bewegungen in das Modell näher ausgeführt werden.

### Umsetzung von Bewegungen

Wie in Abschnitt 3.2.3 ausgeführt ist die Form der Modellteile fest, während sich die Position über die Zeit ändert und von den Eingaben der Komponente, also deren Verhalten, abhängt. Um diese zeitvarianten Bewegungen zu beschreiben, finden die Mover Verwendung. Das in Abbildung 3.12 gezeigte Modell erlaubt die Zuordnung von einem Mover zu mehreren *Movables*, u. a. Parts, Detectors und weiteren Movern. Um eine Kapselung sicherzustellen, kann ein Mover nur mit Movables verbunden werden, die zur gleichen Komponente gehören. Da ein *Mover* selbst ein *Movable* ist, lassen sich diese in Serie schalten, um komplexe Bewegungen zu realisieren. Zudem können andere Komponenten auf der gleichen Hierarchieebene mit Hilfe eines *MoverLink* an einen Mover gebunden werden. Die Details dieser Operation sind in Abschnitt 3.2.5 näher beleuchtet.

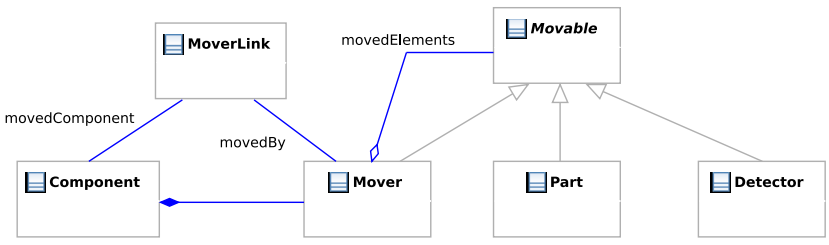



Abbildung 3.12: Meta-Modell für die Umsetzung von Bewegungen bzw. Transformationen

Zur Bestimmung der gültigen Bewegung weist die Verhaltensfunktion der Komponente jedem Mover eine reelle Zahl zu. Wie dieser Vorgang erfolgt, wird für die Automaten nachfolgend erklärt. Der Wert kann auch noch vom Umgebungsmodell modifiziert werden (vgl. Abschnitt 3.2.7). Dieser wird bspw. in Folge einer Kollision zu null gesetzt. Die Achse des Movers berechnet auf der Basis dieses skalaren Werts die eigentliche Transformation, die dann im aktuellen Zeitschritt auf das bewegte Element angewandt wird.

### Automatenbasierte Verhaltensspezifikation

Um das Verhalten einer Komponente zu beschreiben, wird eine bestimmte Variante von Automaten eingesetzt. Diese bestehen aus diskreten Kontrollzuständen, Zustandsvariablen und Transitionen, siehe Abbildung 3.13. Kontrollzustände sind durch Transitionen verbunden, die diskreten Ereignisse im System abbilden. Zeit vergeht nur, wenn der Automat in einem Zustand ist, während Transitionen idealisiert keine Zeit verbrauchen.

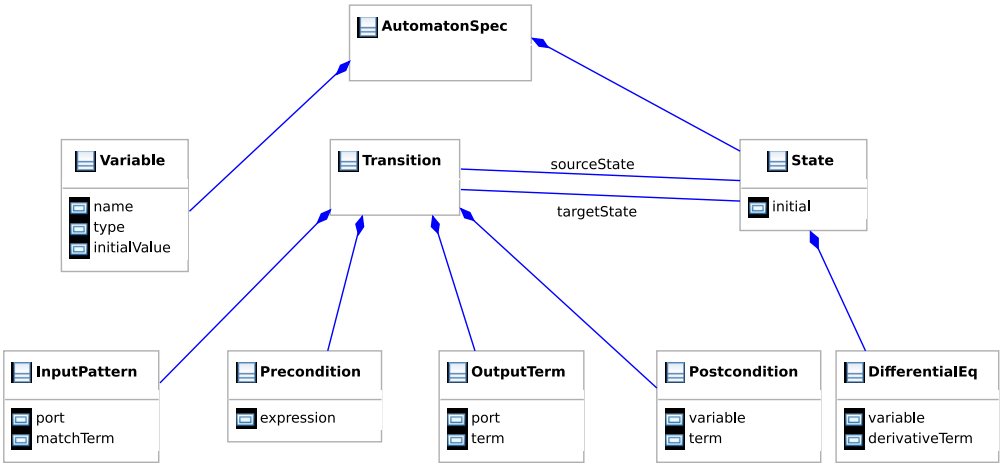



Abbildung 3.13: Meta-Modell der benutzten Automaten

**Grundmodell** Genau ein Kontrollzustand muss als initialer Zustand markiert sein. Ebenso besitzt jede der typisierten Zustandsvariablen eine initiale Belegung. Zur Ausführungszeit ist immer exakt ein Kontrollzustand *aktiv*. Anfangs ist der initiale Zustand aktiviert. Der aktuelle Zustand und die momentane Belegung der Zustandsvariablen werden zusammen als der Zustand des Automaten bezeichnet.

Transitionen verbinden Zustände und können mehrere Eingabemuster (*InputPattern*), Vorbedingungen (*Precondition*), Ausgaben (*OutputTerm*) und Nachbedingungen (*Postcondition*) enthalten. Alle diese (optionalen) Elemente werden mit einer Termsprache beschrieben, die sich an das eingeführte Typsystem anlehnt.

Die Eingabemuster werden an einen Eingabeport der Komponente gebunden und enthalten einen Musterterm (ähnlich zu denen, die bei der Funktionsdefinition im Typsystem verwendet werden), der gültig für den Typ des Ports sein muss. Der Musterterm darf auch Variablen

enthalten. Die verwendete Notation lehnt sich an CSP [Hoa85] an und nutzt ein Fragezeichen, um dem Portnamen vom Musterterm zu trennen. Das Fehlen des Musterterms deutet auf die Abwesenheit einer Nachricht hin (genauer auf die $\epsilon$-Nachricht). Falls der Musterterm Variablen enthält, wird deren Belegung verwendbar für die anderen Elemente der Transition.

Die Vorbedingungen sind boolesche Ausdrücke, die von den Zustandsvariablen und den Variablen aus den Mustertermen der Eingaben abhängen können. Ausgaben werden zum Versenden von Nachrichten genutzt und bestehen aus dem Namen des Ausgabeports, gefolgt von einem Ausrufezeichen und dem Ausdruck, der nach der Auswertung verschickt werden soll. Die Nachbedingungen definieren für jede Zustandsvariable einen Term, der ihren neuen Wert festlegt. Diese werden als Zuweisung mit Hilfe des Gleichheitszeichens geschrieben. Die in den Ausgaben und Nachbedingungen verwendeten Terme dürfen von Zustandvariablen und den in den Eingabemustern vorkommenden Variablen abhängen und müssen typkorrekt sein.

Die Auswertung des Automaten wird am besten schrittweise beschrieben. In jedem Simulationsschritt werden alle Transitionen betrachtet, die den aktiven Zustand verlassen. Eine Transition heißt *bereit*, wenn für jeden referenzierten Eingabeport der zugehörige Musterterm zum aktuell anliegenden Wert passt und alle Vorbedingungen zu *true* ausgewertet werden. Aus der Menge der bereiten Transitionen wird nichtdeterministisch (im Falle des Simulators zufällig gleichverteilt) eine davon ausgewählt. Für diese Transition werden die Ausgaben ausgewertet, verschickt und über die Nachbedingungen der Wert der Zustandsvariablen geändert. Abschließend wird nur der aktive Zustand auf den Zielzustand der Transition gesetzt. Wenn für einen Ausgabeport keine Ausgabe angegeben wurde, wird die leere Nachricht $\epsilon$ (im Falle eines Signalports) oder der Wert aus dem vorherigen Takt (im Falle eines Zustandsports) verschickt. Dergleichen behalten Zustandsvariablen, die in der Nachbedingung nicht vorkommen, ihren Wert. Falls zu einem Zeitpunkt keine Transition für die aktuellen Eingaben bereit ist, ändert sich der aktive Zustand nicht und alle Ausgabeports und Zustandsvariablen verhalten sich, als wenn eine leere Transition ausgeführt worden wäre.

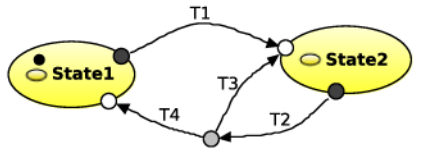
**Pseudohybride Erweiterung** In reinen Softwaresystemen ist häufig nur die relative zeitliche Abfolge von Ereignissen und deren Abhängigkeit untereinander von Interesse. Bei mechatronischen Systemen spielt die tatsächlich vergangene Zeit oft eine entscheidende Rolle, da der Fortschritt physikalischer Prozesse und Bewegungen von der Zeit abhängen.

Um diesem Umstand Rechnung zu tragen, wurden Ideen aus den hybriden Automaten übernommen. Hierbei können Kontrollzustände mittels Differentialgleichungen ergänzt werden, die sich auf die Zustandsvariablen und die Variablen, die den Movern zugeordnet sind, beziehen können. Da dem vorliegenden Ansatz ein diskretes Zeitmodell zu Grunde liegt, lassen sich, anders als in echten hybriden Systemen mit einem kontinuierlichen Zeitmodell, die Differentialgleichungen nicht fehlerfrei auswerten. Die Gleichungen sind eher als syntaktische Erweiterung zu sehen, bieten aber auch die Möglichkeit, eine klare konzeptuelle Trennung zwischen diskreten und kontinuierlichen Änderungen auszudrücken. Die Auswertung der Differentialgleichungen erfolgt diskret, also numerisch. Hierfür werden zu jedem Zeitschritt vor der Auswahl und Ausführung einer bereiten Transition alle Zustandsvariablen entsprechend den Differentialgleichungen im aktiven Zustand aktualisiert.

**Integration mit dem Geometriemodell** Bis zu diesem Punkt haben die geometrischen Modellteile keine Rolle in der Beschreibung der Automaten gespielt. Diese Lücke soll nun geschlossen werden. Der erste Bezug ist die Reaktion des Automaten auf Kollisionen im Modell, genauer gesagt die räumliche Erkennung von Parts in einem Detector. Im Automaten wird das derart realisiert, dass jeder Detector auf eine (schreibgeschützte) boolesche Variable abgebildet wird, die dann in den Transitionsbedingungen genutzt werden kann. Diese Variable ist *true* im Falle von erkannten Parts (also einer Kollision mit dem Volumen des Detectors) und sonst *false*.

Der andere Bezug wird über die Mover hergestellt. Wie schon in Abschnitt 3.2.4 erläutert basiert die Transformation, die ein Mover verwendet, auf einem skalaren Wert, der vom Automaten bestimmt wird. Dieser Skalar, der den Umfang der Bewegung bestimmt, wird als Variable in den Automaten eingeblendet, die ausschließlich in den Differentialgleichungen der Kontrollzustände verwendet werden kann.

**Graphische Syntax** Die für Automaten verwendete grafische Syntax orientiert sich an den mittlerweile weit verbreiteten Zustandsgraphen. Abbildung 3.14 zeigt ein Beispiel für einen Automaten mit Zustandsgraph und Transitionsbedingungen. Die Kontrollzustände sind durch die gelben Ovale dargestellt, der initiale Zustand ist durch einen schwarzen Punkt markiert. Die einzige Zustandsvariable _v wird durch die Angabe von Typ, Name und initialer Belegung deklariert.




(a) State graph

| | | |
|---|---|---|
| **T1** | **in:** | run?**true**; ready?_X |
| | **pre:** | _X > 5 && _d |
| | **out:** | state!**true**; vel!_v+1 |
| | **post:** | _v = _v+1 |
| **T2** | **post:** | _v = _v+1 |
| **T3** | **pre:** | _v > 7 |
| **T4** | **pre:** | _v > 12 |

(b) Transition specification

Abbildung 3.14: Grafische Syntax für Automaten

Die Transitionen sind als gebogene Pfeile gezeichnet, die die Zustände verbinden. Für jede Transition werden Eingabemuster (in), Vorbedingungen (pre), Ausgaben (out) und Nachbedingungen (post) in einer Tabelle aufgeführt. Leere Bedingungen können dabei entfallen. Die Transitionen verbinden die Zustände nicht direkt, sondern über sog. *Verbindungspunkte*, die zu den Zuständen gehören (kleine Kreise). Die Schwarzen sind herausführende, die Weißen sind hineinführende Punkte. Die grauen Punkte gehören zu keinem Zustand und werden als *lokale Verbindungspunkte* bezeichnet. An dieser Stelle werden nur die Letzteren erklärt, da die beiden anderen Arten nur für hierarchische Automaten relevant sind, die hier nicht näher ausgeführt werden. Lokale Verbindungspunkte sind eine syntaktische Erweiterung, um Redundanzen in den Transitionsbedingungen zu vermeiden. Sie werden so interpretiert, dass jede (schleifenfreie) Folge von Transitionen, die durch lokale Verbindungspunkte verknüpft

sind, wie eine einzige Transition behandelt wird, indem alle Transitionsbedingungen zusammengefasst werden. Im aufgeführten Beispiel hätte man statt der Transitionen *T2*, *T3* und *T4* auch zwei Transitionen *T2T3* von *State2* zu sich selbst und *T2T4* von *State2* zu *State1* mit folgenden Bedingungen schreiben können:

| **T2T3** | **pre:** | _v > 7 |
|---|---|---|
| | **post:** | _v = _v+1 |

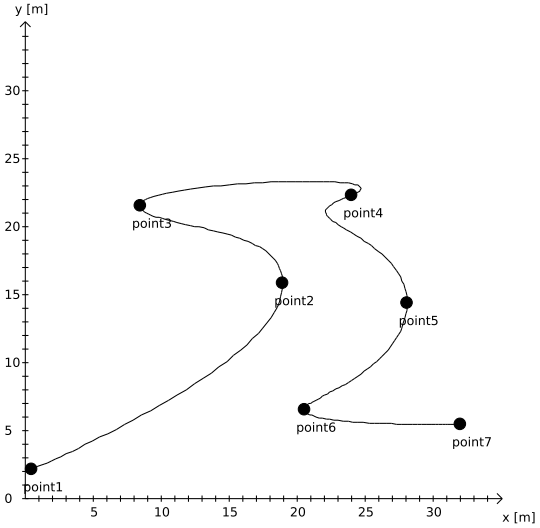
| **T2T4** | **pre:** | _v > 12 |
|---|---|---|
| | **post:** | _v = _v+1 |

### Alternative Spezifikationstechniken

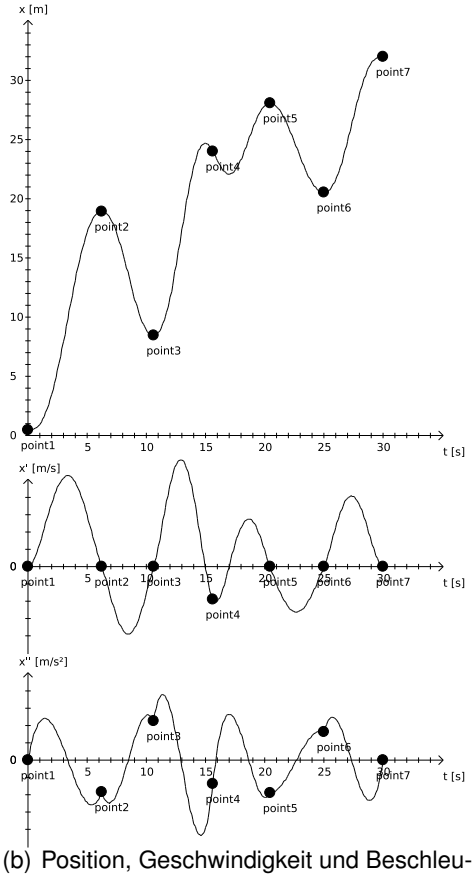
Neben den Automaten unterstützt das Modell auch die Nutzung alternativer Spezifikationstechniken. Während Automaten die mächtigste angebotene Technik darstellen, sind andere Methoden in bestimmten Fällen effizienter. Im Folgenden werden zwei mögliche Alternativen vorgestellt.

**Graphical Function Specification** Bestimmte funktionale Zusammenhänge lassen sich sehr einfach aufzeichnen, aber nur schwer durch Formeln fassen. Ein Beispiel dafür sind Verfahrkurven, wie sie im Anlagenbau häufig zum Einsatz kommen. Eine solche Kurve ist in Abbildung 3.15(a) zu sehen. Die Herausforderung beim Entwurf liegt darin, dass die Geschwindigkeit und insbesondere die Beschleunigung auf beiden verwendeten Achsen innerhalb gewisser Grenzen liegen sollten, um den Verschleiß zu reduzieren und eine hohe Genauigkeit zu gewährleisten. In den Abbildungen 3.15(b) und 3.15(c) sind die Positions-, Geschwindigkeits- und Beschleunigungskurven für die beiden Achsen separat aufgezeichnet. Während die Beschleunigungswerte für die y-Achse durch geeignete Parameterwahl niedrig sind, weisen die der x-Achse hohe Werte auf.

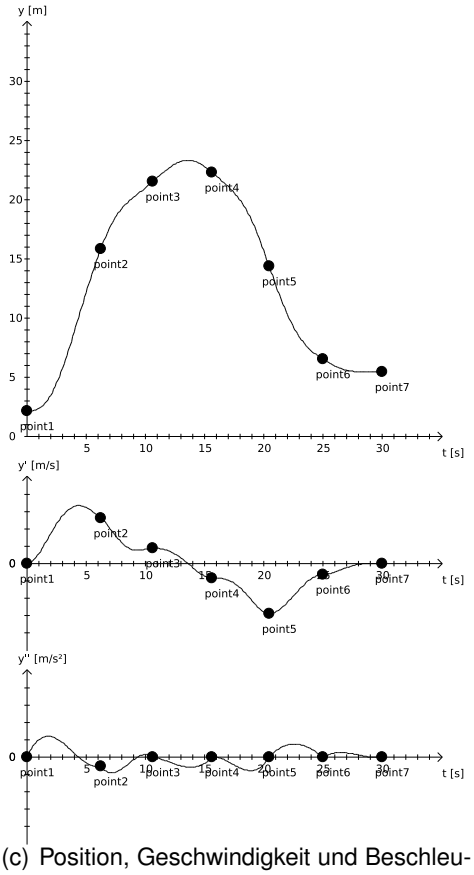
Um die Planung solcher Verfahrkurven zu unterstützen, wird eine grafische Funktionsdefinition angeboten. Eine Kurve wird dabei durch Stützpunkte festgelegt, in denen neben der Position zur entsprechenden Zeit auch die Gewindigkeits- und Beschleunigungswerte im Stützpunkt angegeben werden können. Die fehlenden Zwischenwerte werden durch Polynominterpolation bestimmt. Da nicht nur die X-Y-Kurve, sondern auch alle anderen Kurven aus Abbildung 3.15 automatisch abgeleitet werden können, lassen sich die Auswirkungen von Parameteränderungen in einem geeigneten Werkzeug unmittelbar nachverfolgen.




(a) X-Y-Kurve




(b) Position, Geschwindigkeit und Beschleunigung in X-Richtung




(c) Position, Geschwindigkeit und Beschleunigung in Y-Richtung

Abbildung 3.15: Beispiel für eine grafische Funktionsbeschreibung

**Operator Panels** Bedienbilder werden benutzt, um die Interaktionsmöglichkeiten für den Mensch festzulegen. Die Spezifikation bildet dabei die Elemente der Schnittstelle auf die Ports der Komponente ab.

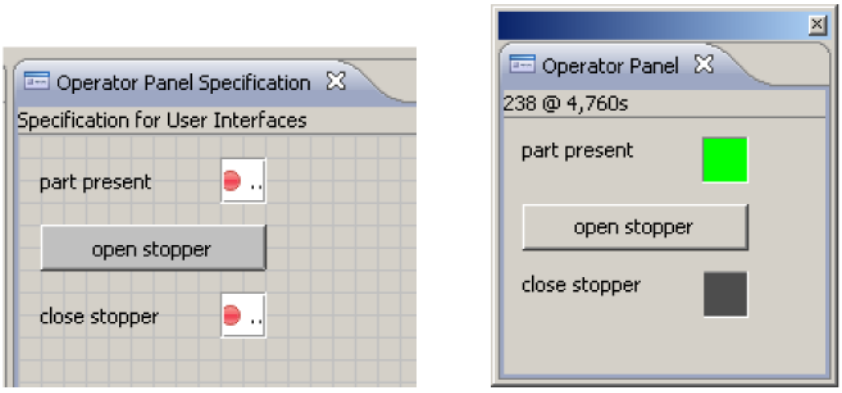
Abbildung 3.16(a) zeigt ein Beispiel für ein einfaches Bedienbild. Die Schaltfläche mit dem Titel *open stopper* wird durch die Terme, die sie mit dem Port *open* der entsprechenden Komponente verbinden, ergänzt:

| | |
|---|---|
| **pressed:** | open!**true** |
| **released:** | open!**false** |

Dabei wird die gleiche Syntax wie für die Automaten verwendet, d. h. die Komponente verschickt das Signal *true* auf dem Port *open*, wenn die Schaltfläche gedrückt wird und sendet *false* während des Loslassens. Das kleine Quadrat repräsentiert eine Signalleuchte, die den aktuellen Zustand eines Ports darstellen kann. Die obere der beiden Leuchten wird durch folgende Terme ergänzt:

| | | |
|---|---|---|
| close_stopper?**true** | → | green |
| close_stopper?**false** | → | grey |

Die Leuchte erscheint damit grün, wenn der Port *close_stopper* das Signal *true* empfängt und ist andernfalls grau. Der Hauptgrund für die Einbindung von Bedienbildern in das Anlagenmodell liegt darin, dass dieses während der Simulation des Modells benutzt werden kann, um mit der Simulation zu interagieren (vgl. Abbildung 3.16(b)).




(a) Spezifikationsansicht (b) Simulationsansicht

Abbildung 3.16: Spezifikation mittels Bedienbildern

#### Verhaltensmodifikation

Wie eingangs erwähnt unterstützt das Modell auch die Verhaltenserweiterung bzw. -modifikation. Dies wird einerseits genutzt, um ergänzende Spezifikationen zu definieren, die den Materialfluss beschreiben, siehe Abschnitt 3.2.6. Zum anderen bildet dies die Grundlage für die in AP3 erarbeitete Fehlermodellierung.

### 3.2.5 Komposition

Unter Komposition wird die kontrollierte Verbindung von mehreren Komponenten zu einer übergreifenden Einheit verstanden. Hierdurch bleibt auch die Modellierung komplexer Systeme beherrschbar, da umfangreiche Problemstellungen in kleinere Teile aufgetrennt werden können. Die Einzellösungen können dann wiederum zusammengefasst werden. Zudem lassen sich auf diese Art und Weise bereits definierte Komponenten einfach in einem neuen Zusammenhang wiederverwenden.

#### Komposition über den Datenfluss

Um Datenflussbeziehungen zwischen Komponenten auszudrücken, werden ihre Ports über Kanäle (vgl. *channel* in Abbildung 3.11) verbunden. Ports dürfen nur gekoppelt werden, wenn sie vom gleichen Typ sind. Zudem können Eingangsports nur mit maximal einem eingehenden Kanal in Beziehung stehen. Ausgangsports dagegen können zu mehreren herausgehenden Kanälen führen, was im Wesentlichen einem Kopieren der verschickten Daten entspricht.

Das Meta-Modell aus Abbildung 3.11 besitzt eine Aggregation namens *subcomponents*, die es erlaubt, eine Komponente durch eine Menge anderer Komponenten zu beschreiben. Eine Komponente darf *entweder* über die Komposition durch andere Komponenten *oder* durch eine Verhaltensspezifikation (wie z. B. Automaten) beschrieben werden. Die Verhaltenserweiterung ist jedoch auch ergänzend zur Komposition möglich. Die „Verdrahtung“ der Unterkomponenten erfolgt, wie bereits beschrieben, mit Hilfe von Kanälen. Zudem können die Ports der umgebenden Komponente den Ports der Unterkomponenten über Kanäle zugeordnet sein. Somit ergibt sich die syntaktische Schnittstelle einer zusammengesetzten Komponente als Teilmenge der Ports der Unterkomponenten sowie aus all deren Parts, Detectors, Bindings und Movers. Die Semantik der Komposition lässt sich induktiv über die Zeit durch kopieren der Datenwerte entlang der Kanäle beschreiben. Eine formale Definition der Komposition findet sich in [Hum09].

Die grafische Syntax basiert auf der aus Abbildung 3.10 bekannten Darstellung der Komponenten als Rechtecke. Wie in Abbildung 3.17 verdeutlicht sind Kanäle als Linien mit Pfeilspitze dargestellt. Die vier Ports, die auf den Hintergrund gezeichnet sind und nicht auf dem Rand einer Komponente liegen, sind die der zusammengesetzten Komponente.

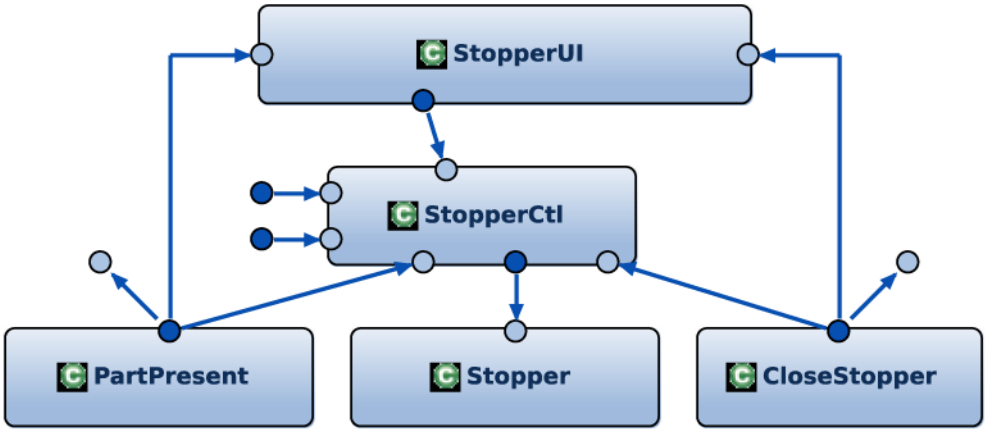



Abbildung 3.17: Grafische Syntax für die Komposition über den Datenfluss

#### Kinematische Komposition

Eine zusammengesetzte Komponente kann zusätzliche Parts einführen und feste Transformationen für ihre Unterkomponenten vorsehen. Zudem kann sie im Rahmen der Komposition mit dem Mover einer benachbarten Komponente verbunden werden. Grafisch wird dies durch die orange, gestrichelte Linie angedeutet, wie Abbildung 3.18 zeigt. Die Linie verbindet eine Komponente mit einem Mover, der durch das Icon mit dem Doppelpfeil dargestellt ist. Im Meta-Modell ist diese Verbindung durch die Klasse *MoverLink* realisiert, die schon in Abbildung 3.12 zu sehen war.

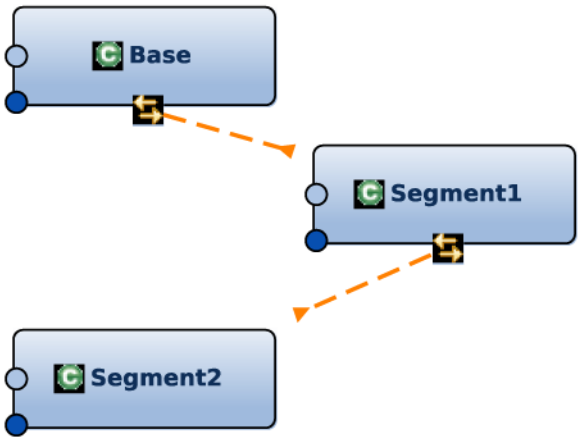



Abbildung 3.18: Grafische Syntax für die kinematische Komposition

### 3.2.6 Materialfluss und generierte Komponenten

Ein wesentlicher Bestandteil des Verhaltensmodells ist die Behandlung von Material und dessen Interaktion mit den Komponenten des Systems. Für die Beschreibung des Materials wird ebenfalls die komponentenbasierte Modellierungstechnik genutzt. Ein Materialelement

ist also auch als Komponente darstellbar. Im Gegensatz zu den Komponenten , die die Anlage beschreiben, kann das Material beliebig oft generiert oder gelöscht werden, d. h. es existieren mehrere Instanzen zur Simulationszeit. Das Einbringen und Entfernen von Material wird über *Entries* und *Exits* realisiert. Der zweite, wichtige Bestandteil eines Materialflussmodells ist die Beschreibung der Interaktion mit den Anlagenkomponenten. Diese Wechselwirkungen werden über die Bindings im Modell hinterlegt.

### Entries und Exits

Der Materialfluss wird über die Verhaltenserweiterung *MaterialFlowSpec* definiert, die auch in Abbildung 3.19 als zentrales Element gezeigt wird. Die Schnittstellen werden durch *Entries*, die dem Einbringen von neuem Material in die Simulation dienen, und *Exits*, die zum Löschen verwendet werden, beschrieben. Beide haben eine geometrische Repräsentation in Form eines Quaders, der den Ort definiert, wo das Material erscheint bzw. sich verliert. Jedem *Entry* wird der Name der Komponente zugeordnet, die von ihm erzeugt wird. Der *Exit* beeinflusst nur das Material, das mit diesem kollidiert.

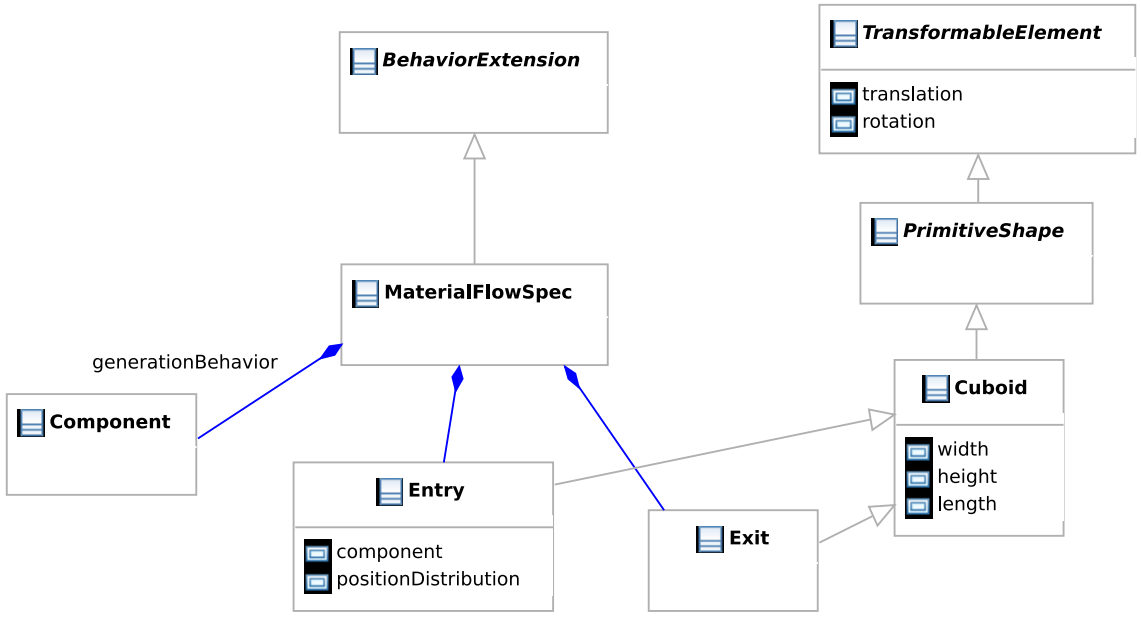



Abbildung 3.19: Meta-Modell für die Definition des Materialflusses

Die Zeitpunkte zum Generieren oder Löschen werden über zwei Komponenten (Assoziation *generationBehavior*) definiert. Die Komponente mit Namen *Assumption* besitzt für jeden Entry einen zugeordneten booleschen Ausgabeport. Wenn immer an diesem Port das Signal *true* anliegt, wird eine neue Instanz der Materialkomponente erzeugt. Deren Position orientiert sich dabei an der räumlichen Ausrichtung des Quaders, der dem Entry zugeordnet ist. Die zweite Komponente namens *Guarantee* hat für jeden Exit einen Ausgangsport. Das

Material wird nur gelöscht, wenn das Signal *true* am entsprechenden Port gesendet wird und sich das Material zu diesem Zeitpunkt innerhalb des dem Exit angehörigen Quaders liegt.

### Bindings

Von großer Bedeutung ist die Interaktion zwischen den Komponenten des Systems und dem Material. Die Erkennung von Stückgut erfolgt mit Hilfe der Detectors und wurde bereits beschrieben. Im Gegensatz dazu wird die Handhabung bzw. die Manipulation über die Bindings geregelt. Diese beschreiben geometrisch die Bereiche, in denen eine Komponente Einfluss auf das Material nehmen kann.

Die Umsetzung im Meta-Modell ist in Abbildung 3.20 zu sehen. Die Bindings werden in den *BindingConditions* genutzt, von denen zwei Unterarten existieren. Das *PortBinding* assoziiert gebundenes Material mit einem Eingangs- oder Ausgangsport der Komponente, während ein *MoverBinding* eine temporäre Verbindung vom Material zu einem Mover herstellt, wodurch die Komponente Einfluss auf dessen Position nehmen kann. Eine Materialkomponente heißt *gebunden* durch eine *BindingCondition*, wenn die Parts des Materials mit der Geometrie aller der Condition zugeordneten Bindings kollidieren. Bei einer Bindung mit einem Mover wird das Stückgut von diesem bewegt. Üblicherweise modelliert ein Binding die Oberfläche eines Transportbands oder die beiden Backen eines Greifers.

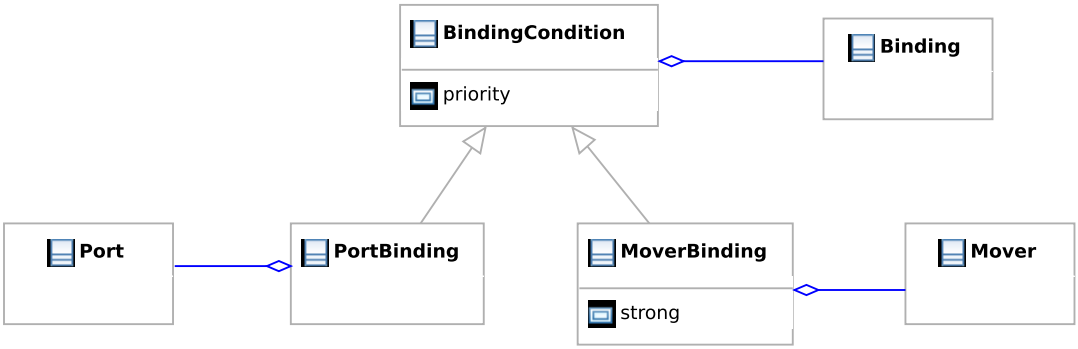



Abbildung 3.20: Meta-Modell zur Definition von Materialbindungen

Wenn Material an einen Port fixiert wird, so ist eine temporäre Vereinigung zischen diesen beiden entstanden. Derartige Bindungen lassen sich bspw. dazu nutzen, um einen RFID-Scanner zu modellieren. Das Binding entspricht dann der Reichweite des Sensors. Das Stückgut kann z. B. einen Port namens *rfid* besitzen, von welchem periodisch der Wert des RFID-Tags gesendet wird. Ist das Bauteil dann in Reichweite des Scanners, verknüpft sich dessen Komponente mit dem Port des Material und liest den entsprechenden Wert aus.

Für beide Arten muss der Fall von gleichzeitigen Mehrfachbindungen behandelt werden. Hierfür dient des Attribut *priority*, mit dem geregelt wird, welche Bindung Vorrang hat. Bei

gleicher Priorität erfolgt eine nichtdeterministische Auswahl (im Falle des Simulators entspricht das einer zufällig gleichverteilten Auswahl).

In der textuellen Syntax werden Ports und Movers mit *BindingConditions* annotiert, indem das Schlüsselwort **bound by** auf deren Deklaration folgt und sich die Namen der benutzten Bindings anschließen. Im Falle eines Movers werden auch die Schlüsselwörter **weakly bound by** und **strongly bound by** verwendet, um den Wert des Attributs *strong* zu kennzeichnen.

### 3.2.7 Kollisionserkennung

Die Geometrie hat einen wesentlichen Einfluss auf die Funktion des Modells, z. B. durch Greifoperationen. Daher wird die Modellsemantik teilweise auch durch eine Kollisionsrechnung ergänzt. Hierfür wird in jedem Simulationsschritt vor der Durchführung geprüft, ob diese zu (nicht erwünschten) Berührungen führt. Falls das der Fall ist, wird die entsprechende Bewegung verhindert, indem die zugehörige Variable auf den Wert Null gesetzt wird. Dieses auf den ersten Blick einfache Modell genügt bereits, um Rückstau und ähnliche Effekte zu simulieren.

### 3.2.8 Beispiel: Förderbänder

Zum besseren Verständnis des Modells wird nachfolgend ein komplettes Beispiel vorgestellt, welches auf zwei Förderbändern basiert. Diese verfügen über Lichtschranken und werden so angesteuert, dass die transportierten Teile am Ende den gleichen Abstand aufweisen.

Die Ausgangssituation lässt sich sehr einfach durch einen Simulationslauf erklären, der in Abbildung 3.21 gezeigt wird. Es ist erkennbar, dass die gelben Materialquader am Anfang des vorderen (roten) Transportbands erscheinen (generiert werden) und dort einen zufälligen Abstand voneinander aufweisen. Am Ende des zweiten (blauen) Bands haben diese dann den gleichen Abstand, der durch eine entsprechende Ansteuerung der beiden Bänder erzeugt wird.

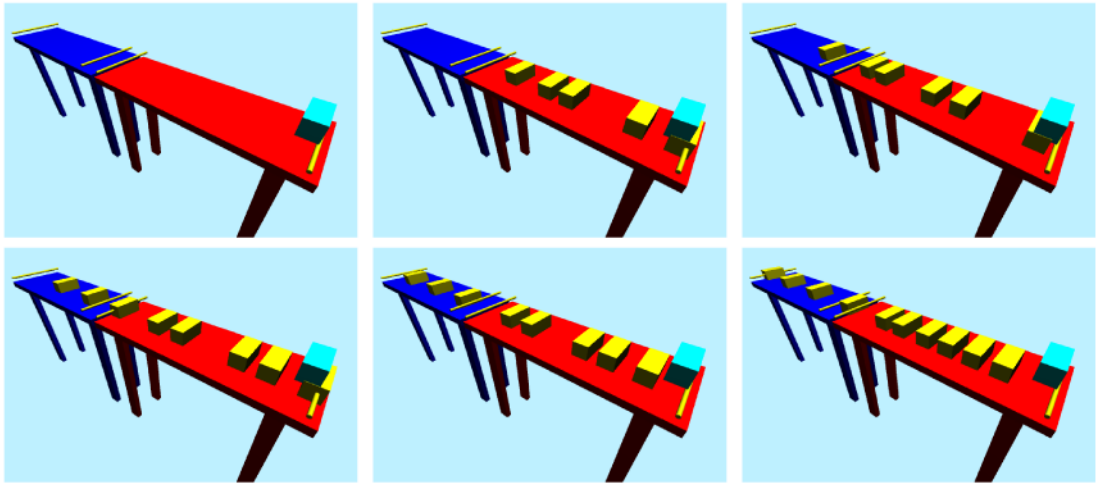


Abbildung 3.21: Mehrere Simulationsschritte des Beispielmodells

Das System besteht aus zwei Förderstrecken, die sich jeweils aus dem Band und zwei Lichtschranken an den jeweiligen Enden zusammensetzen. Beide Einheiten sind gleich aufgebaut und unterscheiden sich nur in der Position (und in der Darstellung in der Farbe). Eine reine Softwarekomponente namens *Controller* dient zur Koordination der beiden Bänder. Dies wird in Abbildung 3.22 verdeutlicht.

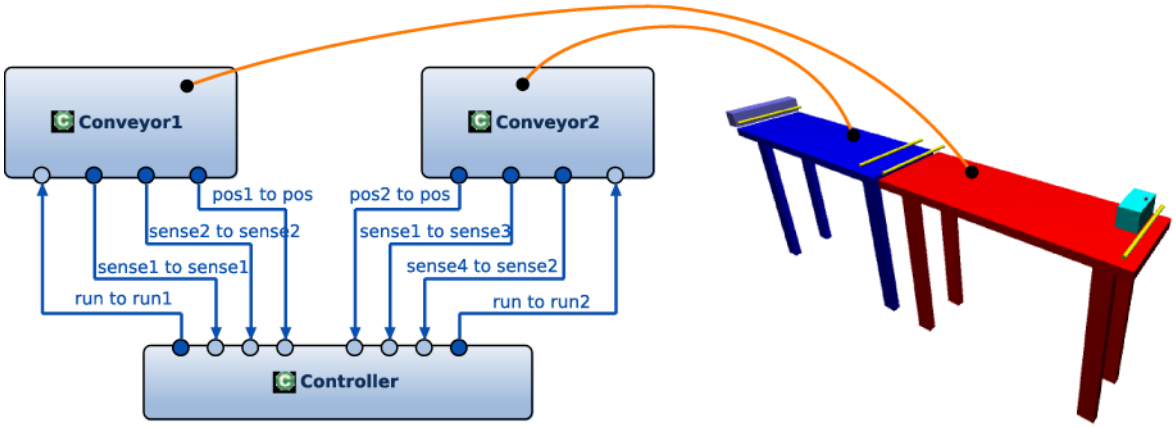



Abbildung 3.22: Komponenten auf oberster Ebene mit entsprechender Geometrie

### Komponente *Conveyor*

Da die beiden Komponenten für die Transportbänder identisch sind, wird hier nur eine beschrieben. Sie besteht aus drei Unterkomponenten: *Belt*, die das bewegliche Band abbildet, *Barrier1* und *Barrier2*, die die beiden Lichtschranken darstellen. Wie in Abbildung 3.23 ersichtlich tauschen diese Komponenten keine Daten untereinander aus, haben aber eine vorgegebene, relative Position. Die Komponente *Conveyor* besitzt vier Ports. Der Eingabeport

*run* nimmt Meldungen vom Typ *boolean* entgegen, die anzeigen, ob das Band laufen oder stoppen soll. Die beiden Ausgänge *sense1* und *sense2* geben die Signale der Lichtschranken weiter, während der Port *pos* den aktuellen Positionswert des Förderbands (bspw. eines Winkelschrittgebers) nach außen führt.

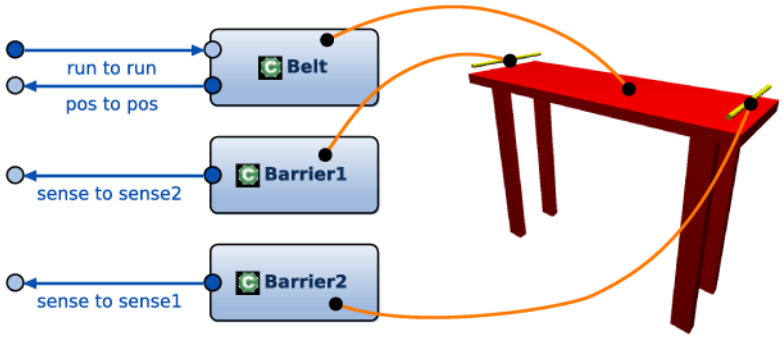



Abbildung 3.23: Komponente *Conveyor* mit zugehöriger Geometrie

**Komponente *Barrier*** Alle Lichtschranken sind gleichartig modelliert, weshalb hier wieder nur das Modell für eine beschrieben wird. Die Komponentenschnittstelle und der definierende Automat sind in Abbildung 3.24 zu sehen. Die Geometrie, die dem Detector *_ray* zugeordnet ist, entspricht dem Zylinder des Lichtstrahls. Der Automat hat nur einen Zustand, der den Kollisionswert an den Ausgang kopiert.

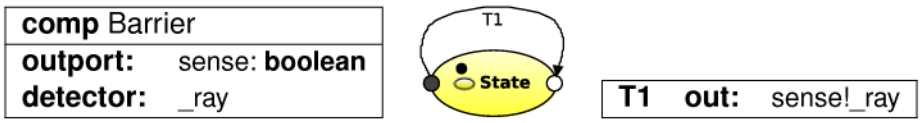



Abbildung 3.24: Komponente und Automat für die Lichtschranke

**Komponente *Belt*** Die syntaktische Schnittstelle der Komponente *Belt* ist in Abbildung 3.25 dargestellt. Der Zweck der einzelnen Ports wurde bereits im Kontext der umgebenden Komponente *Conveyor* erklärt. Das Band selbst wird als Binding modelliert. Das Material, das dieses berührt, wird an den einzigen Mover der Komponente gebunden, der dann die Teile entlang des Bandes transportiert. Die hierfür benutzte lineare Achse ist in der geometrischen Ansicht angedeutet. Die Bindung wurde als schwach gewählt, da ein blockierender Quader das Band nicht zum Stoppen bringen wird, sondern unten durchrutscht. Somit können weitere Quader nachgefördert werden.

| **comp** Belt | |
|---|---|
| **inport:** | run: **boolean** |
| **outport:** | pos: **double** |
| **binding:** | _binding |
| **mover:** | _mover: linear axis |
| | **weakly bound by** _binding |

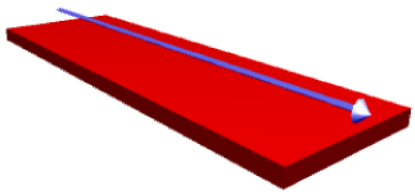
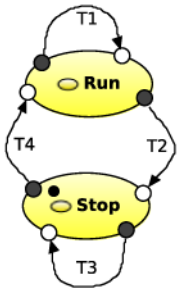


Abbildung 3.25: Syntaktische Schnittstelle der Komponente *Belt*

Das Verhalten wird hier wieder durch einen Automaten beschrieben, der über zwei Kontrollzustände (vgl. Abbildung 3.26) sowie eine Zustandsvariable *_pos* verfügt. Der initiale Zustand *Stop* hält sowohl den Mover als auch die Zustandsvariable konstant (Ableitung gleich null). Der Zustand *Run* dagegen erhöht beide Werte mit einer festen Rate von 0,1 Einheiten pro Sekunde. Alle Transitionen senden den aktuellen Positionswert auf dem entsprechenden Port. Die Zustandsänderungen werden über die auf dem Port *run* empfangenen Werte angestoßen.



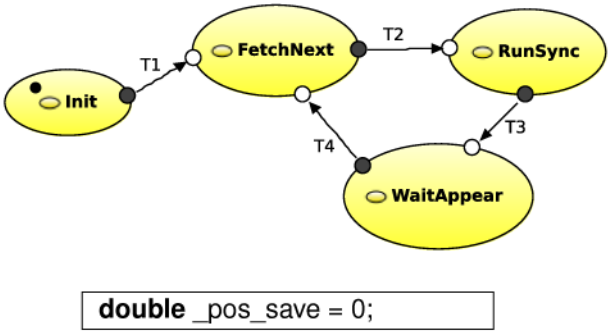

**double** _pos = 0;

| | | |
|---|---|---|
| **Stop** | **deq:** | _mover'=0; _pos'=0 |
| **Run** | **deq:** | _mover'=0.1; _pos'=0.1 |
| **T1** | **in:** | run?**true** |
| | **out:** | pos!_pos |
| **T2** | **in:** | run?**false** |
| | **out:** | pos!_pos |
| **T3** | **in:** | run?**false** |
| | **out:** | pos!_pos |
| **T4** | **in:** | run?**true** |
| | **out:** | pos!_pos |

Abbildung 3.26: Automat für die Verhaltensbeschreibung der Komponente *Belt*

### Komponente *Controller*

Die Komponente *Controller* besitzt sechs Eingangsports: *pos1* und *pos2* vom Typ *double* empfangen die Positionswerte der beiden Bänder, während *sense1*, *sense2*, *sense3* und *sense4* vom Typ *boolean* sind und mit den vier Lichtschranken in Verbindung stehen (über die Schnittstelle der *Conveyor*-Komponenten). Zudem existieren die beiden booleschen Ausgabeports *run1* und *run2*, die die beiden Bänder ein- und ausschalten. Das Verhalten dieser Steuerungskomponente wird über den Automaten aus Abbildung 3.27 definiert. Der Zustand *FetchNext* wartet darauf, dass ein Quader am hinteren Sensor des ersten Transportbandes erkannt wird, um daraufhin das zweite Band zu starten. Im Zustand *RunSync* laufen beide Bänder synchron, bis der Eingangssensor des zweiten Bandes das Material erkennt (Port *sense3*). Sobald das eintritt, wird das erste Band angehalten und der aktuelle Positionswert vom zweiten Band in einer Variablen gespeichert. Der Zustand *WaitAppear* wartet dann, bis die vom hinteren Band gemeldete Position um 0,3 Einheiten höher liegt als der gespeicherte Wert und schaltet daraufhin wieder das erste Band ein und das Zweite aus.




| | | |
|---|---|---|
| **T1** | **out:** | run1!**true**; run2!**false** |
| **T2** | **in:** | sense2?**true** |
| | **out:** | run1!**true**; run2!**true** |
| **T3** | **in:** | sense3?**true**; pos2?_X |
| | **out:** | run1!**false**; run2!**true** |
| | **post:** | _pos_save=_X |
| **T4** | **in:** | pos2?_X |
| | **pre:** | _X − _pos_save > 0.3 |
| | **out:** | run1!**true**; run2!**false** |

Abbildung 3.27: Automat für die Komponente *Controller*

### Generierung von Material

Die Komponente, die das Material repräsentiert, heißt *Brick* und ist trivial aufgebaut, sodass hier keine Details angegeben werden. Sie enthält lediglich ein einzelnes Part, das die Form der Quader beschreibt. Das Verhalten dieser Komponente ist leer, d. h. es wird durch einen Automaten definiert, der einen einzelnen Zustand und keine Transitionen enthält.

In diesem Beispiel erfolgt die Spezifikation des Materialflusses auf der höchsten Hierarchieebene. Diese enthält einen Entry, der am Anfang des ersten Bandes liegt und einen Exit am Ende des zweiten Bandes. Die zugehörigen Quader sind in Abbildung 3.28 ersichtlich, wobei zu beachten ist, dass der Entry als kleiner, roter Würfel dargestellt ist, während der größere, blaue Quader ein Detector ist. Der dem Entry zugeordnete Komponententyp ist *Brick*.

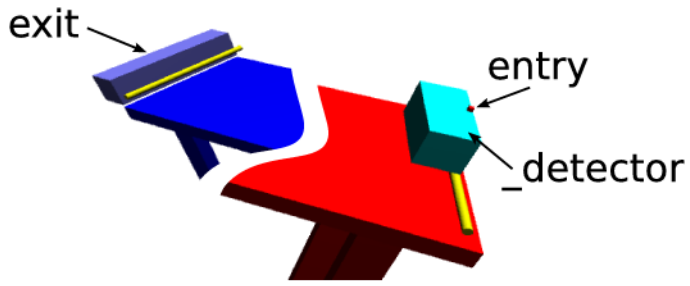
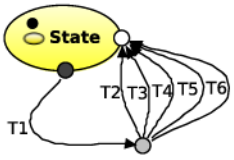



Abbildung 3.28: Detailansicht der für den Materialfluss relevanten Modellelemente

Die Semantik für den Entry, d. h. die Folge von generierten Quadern, wird durch den Automat aus Abbildung 3.29 festgelegt. Dieser besteht aus einem einzigen Kontrollzustand und der Zustandvariable _v1, die als Uhr benutzt wird (daher auch die Differentialgleichung _v1'=1 im Zustand). Die interessante Transition ist *T1*, die den Wert *true* auf den Port schickt, der zum Entry gehört und somit die Generierung eines neuen Stückgutes anstößt. Diese Transition ist nur bereit, wenn sowohl die Uhr den Wert eins erreicht und der Bereich, in dem der Quader erscheinen soll, nicht belegt ist. Letzteres wird über den in Abbildung 3.28 sichtbaren Detectors geprüft: !_detector. Die übrigen Transitionen, die den lokalen Verbindungspunkt

verlassen, setzten die Uhr auf einen Wert zwischen Null und minus vier zurück. Somit vergehen zwischen der Generierung zweier Quader zwischen ein und fünf Sekunden (sofern es keinen Rückstau gibt). Die Transition und somit die Zeit zwischen Generierungsereignissen wird zufällig (bzw. nichtdeterministisch) gewählt, was wiederum bei der Simulation verschiedene Ausprägungen des Materialflusses ermöglicht.




**double** _v1 = 0.9;

| **State** | **deq:** | _v1'=1 |
|---|---|---|
| **T1** | **pre:** | _v1 >= 1 && ! _detector |
| | **out:** | entry_out!**true** |
| **T2** | **post:** | _v1=0 |
| **T3** | **post:** | _v1=−1 |
| **T4** | **post:** | _v1=−2 |
| **T5** | **post:** | _v1=−3 |
| **T6** | **post:** | _v1=−4 |

Abbildung 3.29: Automat zur Beschreibung der Generierungssequenz für Stückgut

## 3.3 AP3: Identifikation qualitätsrelevanter Modellmerkmale und geeigneter Konsistenzbedingungen

Im Rahmen dieses Arbeitspakets wurden zwei unterschiedliche Themenbereiche angegangen:

1. Es wurden Konsistenzbedingungen für die benutzten Artefakte definiert, mit denen sich die Qualität und die Verständlichkeit der Modelle zu jedem Zeitpunkt der Entwicklung messen lassen. Der Schwerpunkt lag hierbei auf der Konsistenz zwischen der Funktionsbeschreibung und den CAD-Modellen. Das Werkzeug AF/STEM wurde um entsprechende Möglichkeiten zur Herstellung von Querverweisen zwischen der Funktionsbeschreibung und den CAD-Modellen (Tracing) erweitert. Zudem wurde ein Konsistenzprüfungsmechanismus ergänzt.
2. In das Funktionsmodell wurde ein Ansatz zur Spezifikation potenziellen Störverhaltens integriert. Dieses kann neben dem Normalverhalten hinterlegt werden. Das Werkzeug AF/STEM wurde im Hinblick auf den Test und die Simulation des Anlagenverhaltens bzw. von Steuerungen unter Fehlerannahmen angepasst.

Im Folgenden werden die Ergebnisse detailliert nach den oben beschriebenen Themenbereichen vorgestellt.

### 3.3.1 Konsistenzprüfung

Es lassen sich die Qualität der isolierten Modelle und der Modelllandschaft unterscheiden. Für Letztere ist vor allem die *Konsistenz* zwischen den verschiedenen Modellkategorien entscheidend. Daher muss die Qualitätssicherung dafür sorgen, dass die im Laufe des Entwicklungsprozesses entstehenden Modelle keine sich widersprechenden Informationen enthalten. Um dies zu gewährleisten wurde das Funktionsmuster um einen Mechanismus für die automatische Konsistenzprüfung erweitert.

#### Konzeption der Konsistenzprüfung

Um die Elemente der abstrakten Funktionsbeschreibung zu den detaillierten Modellen (bspw. aus verschiedenen CAD-Werkzeugen) in Beziehung zu setzen, unterstützt AF/STEM die Erzeugung und die Handhabung von sog. Tracing-Modellen, die die Querverbindungen enthalten. Ein derartiges Modell kann automatisch während des vorwärtsgerichteten Entwicklungsprozesses erzeugt oder im Nachhinein manuell aktualisiert werden. Dadurch dient die Funktionsbeschreibung als zentrales Artefakt in der Modelllandschaft und ermöglicht, Elemente verschiedener Bereiche zueinander in Beziehung zu setzen und abzugleichen. Diese Tracing-Links können für die Dokumentation, das Rational-Management und die Konsistenzprüfung herangezogen werden.

Während des Konsistenzchecks werden automatisch verlinkte Elemente und ihre Eigenschaften verglichen um

- Unterschiede aufzudecken und
- auf nicht verlinkte Elemente hinzuweisen.

Dadurch können bspw. der Typ des benutzten Aktuators oder die Position eines Bauelements im MCAD mit den Repräsentationen aus der Funktionsbeschreibung verglichen werden. Somit lassen sich Divergenzen zwischen den Modellen rechtzeitig aufdecken und beheben, wodurch während der gesamten Entwicklungsdauer die Widerspruchsfreiheit und die Aktualität gewährleistet werden kann.

### Realisierung der Konsistenzprüfung in AF/STEM

Technisch werden die Tracing-Links durch Vergabe von Identifikationsnummern (ID) realisiert. In der Funktionsbeschreibung erhält jedes Modellelement eine eindeutige ID. In den CAD-Modellen wird diese als Attribut hinterlegt bzw. die systeminterne und automatisch vergebene Referenznummer genutzt. Die Elemente aus der Funktionsbeschreibung und den CAD-Modellen werden entsprechend ihrer ID gruppiert und verglichen. Dabei können folgende Fehler aufgedeckt werden:

- Fehlende ID in einem CAD-Element
- Element in der Funktionsbeschreibung oder dem CAD ohne Gegenstück
- CAD bezieht sich auf ungültige ID (z. B. einzelne Zustände aus den Autoamten der Funktionsbeschreibung sollen nicht verlinkt werden)
- Inkonsistenzen innerhalb einer Gruppe

Abbildung 3.30 zeigt den Ergebnisbericht der Konsistenzprüfung zwischen einer Funktionsbeschreibung in AF/STEM und den MCAD- und ECAD-Modellen. Es lässt sich bspw. deutlich erkennen, dass der Aktor für den Hilfsantrieb weder in der Mechanik noch in der Elektrik angelegt ist. Ausgehend von diesen Informationen kann anschließend die Konsistenz durch den Benutzer manuell hergestellt werden. Dieser Weg wurde deshalb gewählt, da bei Fehlern zunächst eine Abstimmung zwischen den Abteilungen notwendig ist. Die Konsistenz der Modelle ist eine Grundvoraussetzung für die Ableitung von Simulationsmodellen (siehe AP5 und 6).

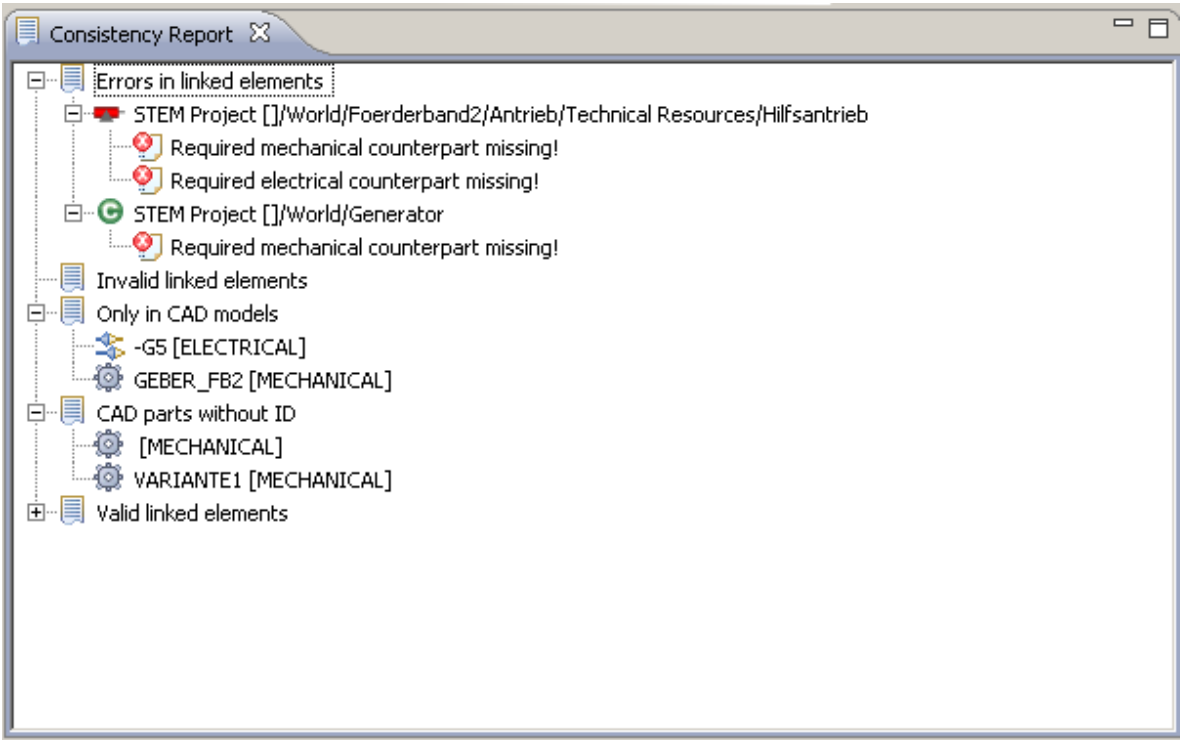



Abbildung 3.30: Konsistenzbericht im Funktionsmuster

### 3.3.2 Fehlermodellierung

Neben dem Testen von Gutabläufen ist die Verhaltensprüfung im Falle einer Störsituation von höchster Bedeutung für die Betriebssicherheit der Maschinen. Jedoch lassen sich solche Zustände für reale Anlagen nur beschränkt und unter erhöhtem Risiko reproduzieren. In der industriellen Praxis wird daher die gezielte Reaktion auf Fehlerszenarien nur für spezielle Bereiche getestet. Dies kann zu Störungen im laufenden Betrieb der Anlage führen. Aus diesem Grund ist eine frühzeitige Berücksichtigung von Maschinenfehlern bereits im Entwurfsstadium sinnvoll. Die VIBN eignet sich besonders für die initiale Qualitätssicherung unter Fehlerannahmen.

In der vorgestellten Funktionsbeschreibung ist eine Fehlermodellierung von Beginn an möglich, siehe Abbildung 3.31. Dabei wird für einzelne Komponenten der Maschine ein alternatives Verhalten definiert, wie bspw. eine defekte Lichtschranke. Zusammen mit der Simulation erlaubt dies die Überprüfung der Anlage auf einzelne oder mehrere Störszenarien. Das fehlerhafte Verhalten lässt sich automatisiert in die nachfolgend generierten VIBN-Modelle integrieren. Zudem können aus den erfassten Daten Testfälle abgleitet werden, die eine möglichst vollständige Prüfung der Anlage erlauben.

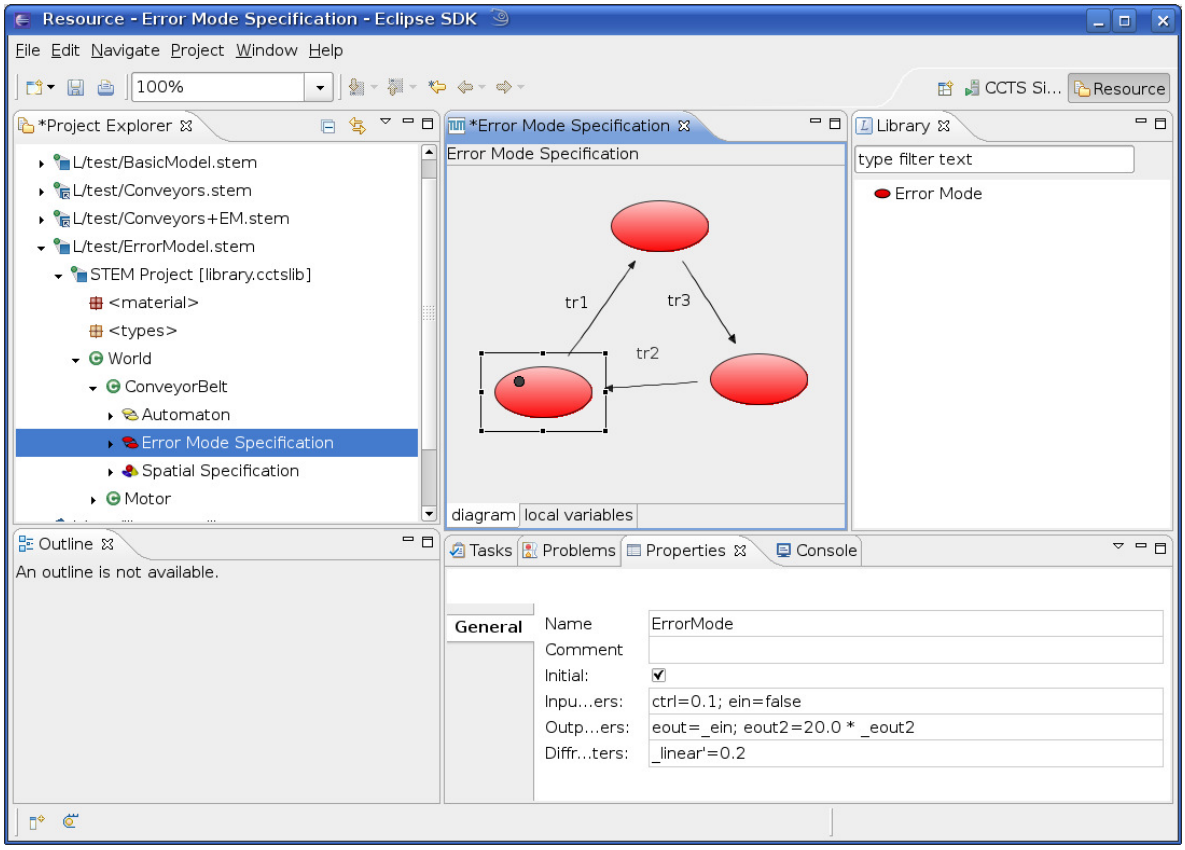



Abbildung 3.31: Realisierung der Fehlermodellierung im Funktionsmuster

### Ziele und Designentscheidungen bei Erweiterung der Funktionsbeschreibung

Das mit der Funktionsbeschreibung verfolgte Ziel ist, das fehlerhafte Verhalten von einzelnen Komponenten spezifizieren zu können, um dadurch die Reaktionen der Steuerungssoftware in Störsituationen zu reproduzieren. Dabei ist die eigentliche Fehlerursache zweitrangig. Es gibt bspw. mehrere Gründe für das Versagen eines Sensors: Spannungsfluktuationen, physischer Defekt, elektromagnetische Felder oder metallische Späne auf dem Messbereich. Der durch die Steuerung beobachtbare Effekt ist jedoch immer von gleicher Natur, der Sensor sendet falsche Daten. Daher liegt die Grundannahme nahe, dass die Steuerung in all diesen Fällen gleich reagieren wird und hier Äquivalenzklassen hinsichtlich des Verhaltens gebildet werden können.

Für den Test und die Simulation unter Störannahmen ist das konkrete (fehlerhafte) Verhalten der Komponenten wichtig, nicht die Wahrscheinlichkeit oder die statistische Verteilung ihres Auftretens. Daher wird das Fehlverhalten mit ähnlichen Mitteln beschrieben, wie sie auch für den Gutablauf zum Einsatz kommen (siehe dazu Abschnitt 3.3.2).

Im Rahmen dieses Projektes liegt der Schwerpunkt auf der Modellierung von Hardwarefehlern, da für die VIBN lediglich das Maschinenmodell ausgeleitet wird. Jedoch auch die

in der Funktionsbeschreibung hinterlegte Steuerungslogik kann von den hier vorgestellten Mechanismen profitieren.

Eine weitere, wichtige Designentscheidung, die an dieser Stelle getroffen wurde, ist die Trennung von Normal- und Fehlverhalten in der Spezifikation. Dies hat folgende Vorteile:

- Einfachere und übersichtlichere Zustandsgraphen
- Getrennte Sichten auf das System, welche jeweils das Normalverhalten und verschiedene Fehlerszenarien beinhalten (sog. *separation of concerns*)
- Änderungen am Normalverhalten sowie die Wiederverwendung von Störmodellen werden besser unterstützt und entscheidend vereinfacht

Die Konsequenz dieser Entscheidung ist eine beschränkte Änderungsmöglichkeit hinsichtlich des Gutablaufs. Es können jeweils nur die Ein- und Ausgaben der Komponente falsifiziert werden, nicht aber interne Berechnungsschritte. Die durchgeführten Fallstudien haben gezeigt, dass diese Maßnahmen für die diskreten Ein- und Ausgaben ausreichend sind.

Von Bedeutung ist zudem die Trennung zwischen dem eigentlichen Fehlverhalten und den auslösenden Bedingungen. Dies wurde derart erreicht, dass Letztgenannte als Übergänge in einem Betriebsmodi-Automaten modelliert und die eigentlichen Abweichungen vom Normalverhalten mit jeweiligen Betriebsmodi assoziiert wurden. Dadurch kann u. a. das Normalverhalten als ein Spezialfall des Fehlverhaltens modelliert werden, nämlich als ein Betriebsmodus, in dem keine Abweichungen stattfinden. Weitere Vorteile dieser Entscheidung sind:

- Möglichkeit dynamisch, d. h. während der Laufzeit, zwischen verschiedenen Fehlermodi und/oder Normalverhalten umzuschalten
- Komplizierte Ausfallszenarien mit Fehler-Logik-Komponenten sind realisierbar
- Einfache Austauschbarkeit und Erweiterbarkeit von Störszenarien

#### Konzeption und Realisierung der Fehlermodellierung

Im Folgenden wird die Architektur zur Erweiterung der Komponenten mit Störverhalten vorgestellt. Diese ist für eine Komponente mit nur einer Eingabe und einer Ausgabe in Abbildung 3.32 schematisch verdeutlicht. Die Dreiecke beschreiben die Ports und die Richtung des Datenflusses. Wie bereits erwähnt, besteht die Fehlerspezifikation einer Komponente aus einem oder mehreren Fehlermodi, welche die Abweichungen vom Normalverhalten festlegen. Jeder Fehlermodus ist durch zwei *Filter* charakterisiert: Der Eingabefilter (im Bild als “I-Filter” bezeichnet) verändert die Eingabe bevor diese durch die Originalkomponente verarbeitet wird und der Ausgabefilter (“O-Filter”) führt die Nachverarbeitung der Ausgabe durch. Von großer Bedeutung ist zudem der Fehlermodus-Automat (im Bild “Error Mode Function”), welcher zu jedem Zeitpunkt den anzuwendenden Störfall angibt. In Abbildung 3.32 ist weiterhin zu sehen, dass die Schnittstelle der Originalkomponente in der Schnittstelle der Erweiterung enthalten ist. Dadurch und durch die Tatsache, dass die Filterung der Ein- und Ausgaben sowie das Schalten des Fehlermodus-Automaten keine (logische) Zeit beansprucht, wird die Erweiterung für die umgebenden Komponenten transparent. Mit Hilfe von zusätzlichen Ports, die durch den Fehlermodus-Automat geschrieben oder gelesen werden

können, besteht die Möglichkeit, das Auswählen des aktuellen Fehlermodus von außen zu beeinflussen und damit globale Ausfallszenarien zu realisieren. Die restlichen Komponenten in Abbildung 3.32 sind sog. Selektoren, welche die Ausgaben des jeweils aktuellen Filters nach außen weiterleiten.

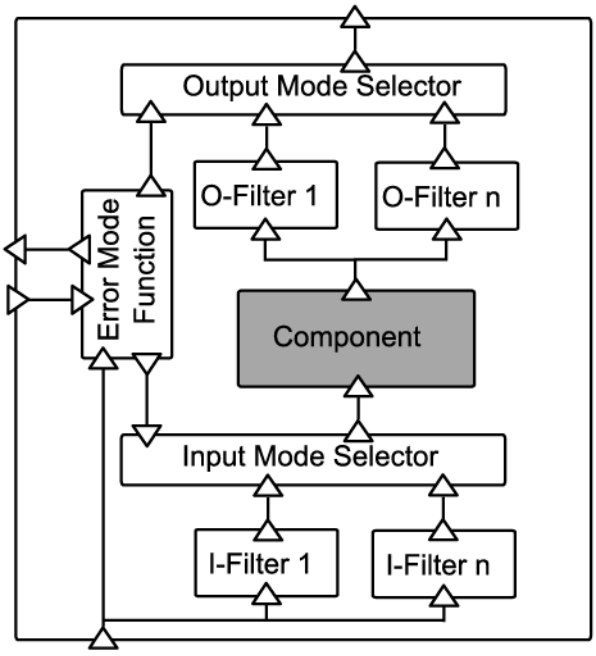



Abbildung 3.32: Architektur der Fehlverhalten-Erweiterung einer Komponente

Die I/O-Filter und der Fehlermodus-Automat müssen vom Benutzer spezifiziert werden. Dagegen sind die Selektoren in die Semantik der Erweiterung integriert. Abbildung 3.33 zeigt die Editoren für die Spezifikation der o. g. Elemente. Der Automat (oben) besteht aus Fehlermodi und verbindenden Transitionen. Er kann als eine Projektion von internen, fehlerhaften Vorgängen auf das beobachtbare Fehlverhalten verstanden werden. So ist bspw. das Durchbrennen einer Sicherung als wahrscheinlich anzunehmen, wenn sie sich auf eine bestimmte Temperatur erhitzt hat. Diese Tatsache kann durch einen Automaten modelliert werden, der erst nach einer bestimmten Zeit den Fehlermodus wechselt. Im Allgemeinen kann der Wechsel des Fehlermodus auf der Basis folgender Ereignisse geschehen:

- Nichtdeterministisch, jederzeit
- Explizit, getriggert von außen durch spezielle Signale
- Basierend auf der Beobachtung der Eingaben der Originalkomponente

Natürlich können auch Kombinationen aus diesen Ereignissen für das Bestimmen des aktuellen Fehlermodus genutzt werden. Der letzte Ereignistyp ist relevant, falls Fehler ausgedrückt werden sollen, die bspw. durch den Verschleiß bedingt werden (nach einer gewissen Anzahl von Aktivierungen) oder die das Schalten des Fehlermodus nur in definierten Zuständen der Komponente erlauben. So kann z. B. ein elektronisches Gerät nur dann störanfällig sein, wenn es eingeschaltet ist. Außerdem können während der Lebensdauer des Systems verschiedene Szenarien sinnvoll sein. Ein Fehler kann irreversibel, z. B. eine durchgebrannte

Sicherung, oder temporär sein, wenn etwa Sensordaten wegen des Überlaufs des Sendepuffers verloren gingen. Im Falle der temporären Anomalien muss die korrekte Wiederaufnahme des Betriebs sichergestellt werden. Da das fehlerhafte Verhalten auf der Basis des Gutblaufs aufbaut, kann der Zustand, in welchem die Komponente in einen Störfall wechselt, sich von dem Zustand unterscheiden, in welchem sie diesen wieder verlässt. Daher sind die Transitionsbedingungen des Fehlermodus-Automaten stark an die Realisierung der Originalkomponente gekoppelt.

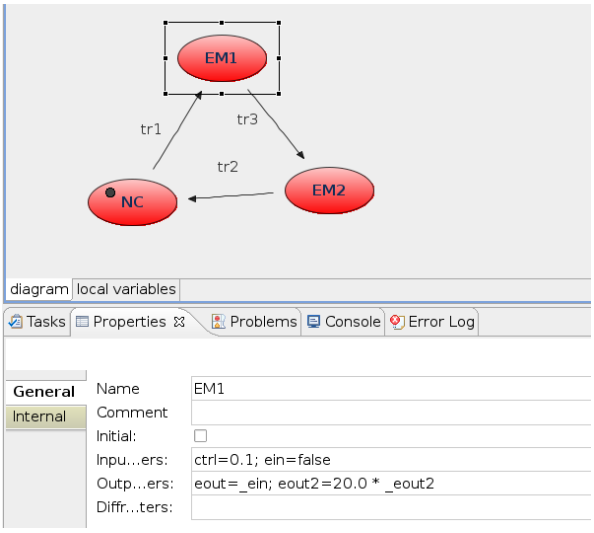



Abbildung 3.33: Fehlermodus-Automat und die Spezifikation eines Fehlermodus im Funktionsmuster

Die I/O-Filter lassen sich dagegen gut strukturieren und von verschiedenen Komponenten wiederverwenden. Dies wird im nächsten Abschnitt genauer ausgeführt. Die Spezifikation des Fehlermodus “EM1” durch den Filter im Funktionsmuster ist in Abbildung 3.33 unten zu sehen. Der Eingabefilter `ctrl = 0.1` setzt den Port *ctrl* auf den konstanten Wert 0,1. Der Ausgabefilter `eout2 = 20.0 * _eout2` gibt auf dem Ausgabeport *eout2* das Zwanzigfache des korrekten Wertes, was durch die spezielle Variable *_eout2* gekennzeichnet ist. Sind die Filterfelder leer (oder alternativ mit `port = _port` spezifiziert), so werden die Ein- und Ausgaben nicht beeinflusst und damit und das Originalverhalten nicht verfälscht.

Ein Simulationszyklus der mit Fehlverhalten erweiterten Komponente ist in Abbildung 3.34 dargestellt. Dieser lässt sich in folgende vier Phasen unterteilen:

1. Eventueller Fehlermode-Wechsel
2. Werte der Eingabeports werden durch den aktuellen I-Filter verfälscht
3. Zustandswechsel des originalen Automaten findet unter Benutzung von bereits verfälschten Eingaben statt
4. Verfälschen der Ausgaben und Sichtbarkeit für die Außenwelt

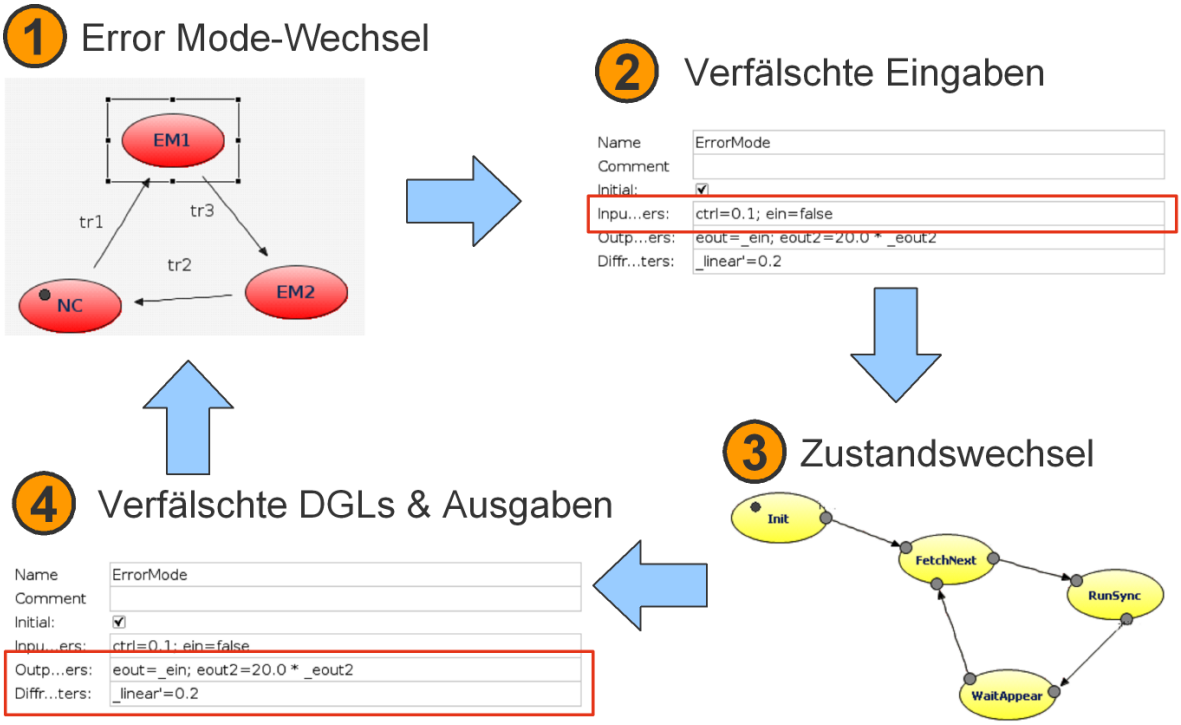



Abbildung 3.34: Ein Simulationszyklus für Komponenten mit Fehlverhalten

### Fehler & Fehlertypen

Die Tabelle 3.1 stellt eine Taxonomie von Fehlertypen vor. Sie sind nach den Komponenten, die üblicherweise in Maschinen und Anlagen vorkommen, gruppiert. Bei den durchgeführten Fallstudien stellte sich heraus, dass es eine begrenzte Anzahl verschiedener, beobachtbarer Fehlerszenarien und dadurch auch Filtertypen gibt:

- $n$-*delay*-Filter, die die Werte um $n$ Simulationsschritte verzögert weiterleiten
- 0-Filter, die keine Werte durchlassen
- *const*-Filter, die immer den gleichen Wert ausgeben
- *id*-Filter, die keine Verfälschung vornehmen

Die letzte Sektion der Tabelle zeigt, dass der gleiche Mechanismus auch für die Softwaremodule anwendbar ist. Das ist vor allem dann sinnvoll, wenn die Softwaresteuerung auf mehrere Komponenten verteilt ist und diese aber isoliert getestet werden sollen.

Bisher wurde lediglich die lokale Fehlererweiterung von Hardwarekomponenten präsentiert. Oft bestehen aber technische Systeme aus mehreren, interagierenden und parallel ausführenden Komponenten. Eine wichtige Herausforderung ist dabei, *komplexe* Ausfallszenarien modellieren zu können, in denen mehrere Komponenten ein vom Normalfall abweichendes Verhalten zeigen.

| Komponente | Fehlerart | Fehlermodell |
|---|---|---|
| Motor | steht | in-/0-out-filter |
| | falsche Geschwindigkeit | dg-filter |
| | falsche Richtung | in-filter |
| Sensor/Lichtschranke | Signalausfall | 0-in-/0-out-filter |
| | Wertfehler | in-/out-filter |
| | Rauschen | in-/out-filter |
| | Jitter: zu früh/zu spät | out-delay-filter |
| Fließband | steht | in-filter |
| | falsche Geschwindigkeit | dg-filter |
| | falsche Richtung | in-filter |
| SW-Steuerung | Wertfehler | in-/out-filter |
| | Jitter: zu früh/zu spät | out-delay-filter |

Tabelle 3.1: Fehlerarten und -modelle

Ein komplexes Szenario bezieht mehrere Komponenten ein, welche ein Fehlverhalten

- simultan,
- in einer festgelegten Reihenfolge und/oder
- auf bestimmte Weise zeitversetzt

aufweisen. Für die Organisation komplexer Szenarien sieht die vorgestellte Fehlererweiterung zusätzliche Ein- und Ausgabeports vor, deren Werte den Fehlermoduswechsel beeinflussen bzw. die Informationen über den aktuellen Fehlermodus der Außenwelt zur Verfügung stellen. Dadurch können die verschiedenen Fehlermodus-Automaten ihre Aktionen direkt synchronisieren oder durch eine zentrale Koordinationskomponente angesteuert werden.

#### Verwendungsszenarien

Die ausgeführten Ergebnisse sind für die nachfolgend beschriebenen Anwendungsszenarien nützlich und wurden zum Teil auch bereits erfolgreich eingesetzt.

Ein naheliegendes Einsatzgebiet ist die Simulation der mit Fehlern erweiterten Modelle in der Funktionsbeschreibung. Dies ist möglich, da sowohl die Funktionsbeschreibung als auch ihre mit Fehlern erweiterte Version ausführbar sind. Der Ansatz kann dazu dienen, das Verhalten des Gesamtsystems besser zu verstehen, was besonders für die Validierung von Nutzen ist, wenn bspw. die Gesamtlösung den Kunden demonstriert werden soll. Die Simulation kann auch für das Testen der Spezifikationen auf der Modellebene genutzt werden. Durch ad-hoc erzeugte oder vorgefertigte Eingaben, kann der Benutzer entweder manuell oder unter Verwendung spezieller Testorakel überprüfen, ob das System sich wie erwartet verhält und nicht gefährliche Maschinenzustände einnimmt.

Ein weiterer, wichtiger Bereich ist die Verwendung der mit Fehlern erweiterten Spezifikation im Rahmen der VIBN für das Testen von Steuerungssoftware. Somit kann die Reaktion der Steuerung auf kritische Situationen abgeschätzt und das Risiko für den Betrieb der Anlage reduziert werden. Zudem lassen sich dadurch die Kosten für die Fehlerbeseitigung verringern.

Die FMEA (*Failure Modes and Effects Analysis*) ist eine weitverbreitete Methode, um mögliche Störzustände (engl. failure modes) zu identifizieren und ihre Auswirkungen auf des System vorherzusagen. Die vorgestellte Technik für die Fehlermodellierung kann eine FMEA bei der Analyse unterstützen. Während der Simulation sind die Folgen direkt beobachtbar.

Der vorgestellte Ansatz kann zudem für die formale Verifikation und die Testfallgenerierung genutzt werden.

## 3.4 AP4: Auswertung der Datenstruktur der verwendeten CAD-Systeme

In AP 4 wurde der Datenaustausch zwischen dem Funktionsmodell und den eingesetzten CAD-Systemen näher untersucht. Die Zielstellung hierbei war, einen Ansatz zu entwickeln, der herstellerunabhängig einen Informationstransfer von der Konstruktionsebene hin zur Funktionsbeschreibung ermöglicht. Hierzu wurden unterschiedliche Kopplungsvarianten und Datenformate analysiert. Auf der Basis der XML (Extenible Markup Language) wurde ein Austauschformat für CAD-Systeme definiert, welches die formulierten Anforderungen erfüllt.

### 3.4.1 Kopplung von Funktionsbeschreibung und CAD-Systemen

Prinzipiell lassen sich zwei unterschiedliche Kopplungsmöglichkeiten zwischen dem Funktionsmodell und den CAD-Modellen unterscheiden, siehe Abbildung 3.35. Zum einen ist eine direkte Kopplung von Funktionsmodell und CAD-Systemen möglich, was einen Online-Zugriff auf die hinterlegten Informationen gleichkommt. Zum anderen können Daten in einem Zwischenformat gespeichert werden, welches dann vom Funktionsmodell zu interpretieren ist. Letzteres kann auch als Offline-Kopplung bezeichnet werden.

Kopplungsvarianten:

- Direkte Kopplung (online)
- Kopplung über Zwischenformat (offline)

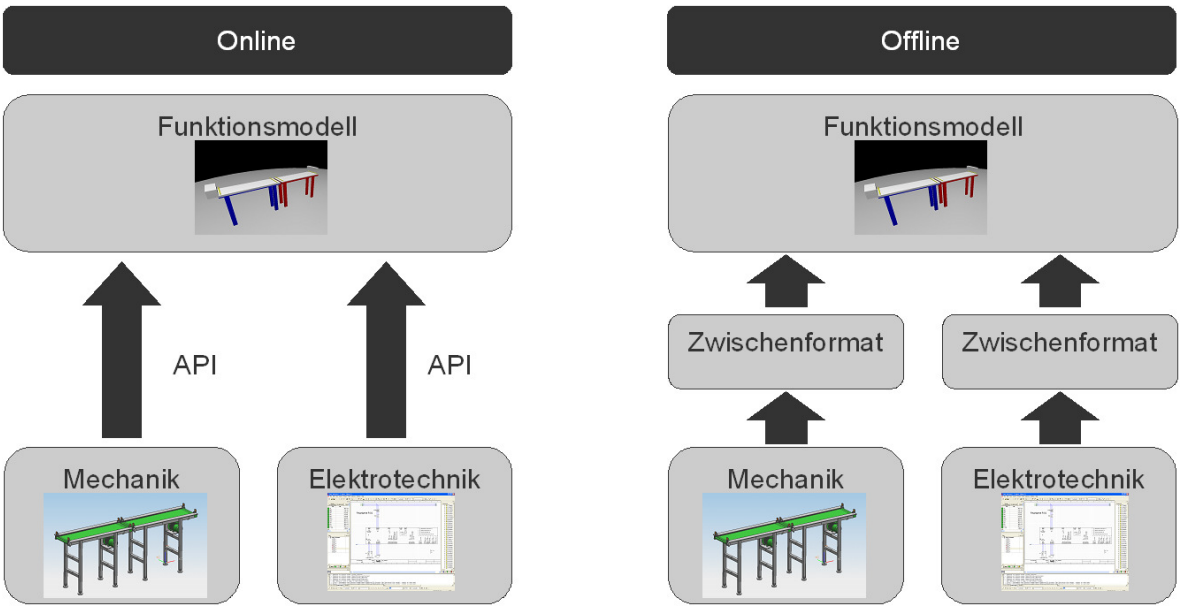



Abbildung 3.35: Kopplungsmöglichkeiten von Funktionsmodell und CAD-Systemen

#### Direkte Kopplung

Moderne CAD-Systeme bieten über Programmierschnittstellen, sog. APIs (Application Programming Interface), einen umfangreichen Zugriff auf das hinterlegte Modell. Dies trifft für Werkzeuge sowohl aus dem Bereich der mechanischen Konstruktion als auch für die Elektrokonstruktion zu. Die Verwendung der vom Hersteller zur Verfügung gestellten Schnittstellen hat den Vorteil, dass beliebige Informationen, soweit von der API unterstützt, aus dem CAD-System extrahiert werden können. Hierdurch ergibt sich eine Unabhängigkeit, die von standardisierten Datenformaten nicht immer gewährleistet werden kann, da dort u. U. nicht alle Teile des Modells hinterlegt sind.

Als Nachteil dieser Lösungsvariante ist die feste Bindung an ein spezifisches Produkt zu sehen. Klassen und Funktionen zur Auswertung der Modelle sind von Hersteller zu Hersteller unterschiedlich. Die Integration in die Funktionsbeschreibung müsste somit jeweils angepasst werden. Eine Austauschbarkeit des CAD-Systems ist dadurch nicht mehr gegeben. Aus diesem Grunde wurde die Online-Kopplung im Rahmen des beschrieben Forschungsvorhabens nicht weiter verfolgt. Der hier aufgegriffene Ansatz soll eine Substitution von CAD-Werkzeugen ermöglichen. Daher wurde im Projekt AutoVIBN der Datenaustausch über ein Zwischenformat realisiert.

#### Kopplung über Zwischenformat

Als Transfermedium wurden unterschiedliche Möglichkeiten betrachtet. Zunächst wurden Standarddatenformate näher untersucht. Hier ist vor allem der *Standard for the Exchange of Product Data* zu nennen, der auch unter dem Akronym *STEP* bekannt und genormt ist [ISO94]. Des Weiteren wurde ein XML-basierter Datenaustausch analysiert.

### 3.4.2 Datenformate

In den folgenden Abschnitten werden unterschiedliche Datenformate für die Anbindung der CAD-Werkzeuge genauer beschrieben.

#### Standarddatenformate

Der STEP-Standard beschreibt nicht nur ein Austauschformat, sondern stellt ein detailliertes Produktmodell dar. Im Maschinen- und Anlagenbau ist dieser weit verbreitet und i.d.R. in allen gängigen MCAD-Werkzeugen implementiert. Hierbei handelt es sich jedoch lediglich um Teil 21 der Norm, in welchem das Klartextformat zum Informationsaustausch beschrieben ist. Die Systemanbieter setzen allerdings weiterhin auf deren proprietären Modelle, wodurch der STEP-Standard nur als Mittel zum Austausch von Geometriedaten zum Einsatz kommt. Darüber hinaus haben die Untersuchungen im Rahmen diese Forschungsvorhabens gezeigt, dass nicht alle benötigten Daten in den STEP-Dateien hinterlegt sind. An dieser Stelle sind z. B. die kinematischen Freiheitsgrade zu nennen, die zwar in der Norm definiert sind

(Teil 105), von gängigen CAD-Systemen aber nicht in den entsprechenden Dateien hinterlegt werden.

Das Forschungsprojekt MechaSTEP [Pav01] hatte die Entwicklung eines neutralen Datenformates zum Ziel, welches vornehmlich zur Simulation mechatronischer Systeme dienen sollte. Hierbei wurde ein zu STEP konformes Austauschformat entwickelt, welches exemplarisch für zwei Mehrkörpersimulationssystem angewendet wurde. Aufgrund der fehlenden Implementierung in MCAD-Werkzeugen, wurde der MechaSTEP-Ansatz hier nicht weiter verfolgt.

Das Format JT (Jupiter Tesselation) ist ein vor allem in der Automobilindustrie häufig verwendeter Standard. Ursprünglich von der Firma UGS (heute Siemens PLM Software GmbH) definiert, steht JT heute quelloffen zur Verfügung. Neben einer Darstellung mit unterschiedlichen Abstraktionsstufen, sog. Levels of Detail (LOD), sind die genauen Daten als BREP (Boundary Representation) hinterlegt.

Im Hinblick auf die in diesem Forschungsprojekt vorliegenden Anforderungen hat sich keines der beschriebenen Austauschformate als geeignet erweisen. Dies ist dem Umstand geschuldet, dass nicht alle benötigten Daten aus den Standardformaten extrahiert werden können. Hier ist als Beispiel STEP zu nennen, obwohl die Kinematik Teil der Norm ist, gehen sämtliche Informationen hierzu beim Export in das Klartextformat i.d.R. verloren. Die Anbieter von CAD-Systemen setzten weiterhin auf proprietäre Modelle, deren Leistungsmerkmale sich nicht unbedingt mit dem STEP-Standard vereinigen lassen [Cui00]. Zudem ist es notwendig, dass für eine systemunabhängige Anwendung, das Austauschformat von den CAD-Werkzeugen unterstützt wird, was bei MechaSTEP nicht gegeben ist.

#### XML-basiertes Austauschformat

Aus den im vorangegangenen Abschnitt genannten Gründen wird im Rahmen des vorliegenden Forschungsvorhabens ein XML-Austauschformat verwendet, das alle benötigten Daten aus den CAD-Systemen beinhaltet. Der Aufbau und die Struktur wurden unter Berücksichtigung der Belange der Projektziele definiert. Durch die Verwendung dieses Austauschformates ist eine Unabhängigkeit von den CAD-Systemen gegeben. Hierbei kann jedes beliebige Werkzeug zum Einsatz kommen, vorausgesetzt, die geforderten Daten können in der XML-Datei abgebildet werden. Zum Schreiben dieser XML-Datei können die Programmierschnittstellen des jeweiligen CAD-Systems verwendet werden. Dies hat den Vorteil, dass meist auf den vollen Umfang des Modells zurückgegriffen werden kann und sich somit mehr Daten extrahieren lassen.

Das definierte XML-Schema ist für die Mechanik und für die Elektrotechnik gültig. Das zentrale Element ist die Komponente (engl. component), die als Strukturierungsmittel dient. Jede Komponente verfügt über einen Namen und eine eindeutige Identifikationsnummer (ID). Hiermit werden die Bauteile im MCAD und die Betriebsmittel im ECAD beschrieben. Informationen, welche die Komponenten näher charakterisieren, werden als Attribute (engl. attribute) selbiger beigefügt. Das hier vorgeschlagene XML-Schema ist in Abbildung 3.36 exemplarisch dargestellt.

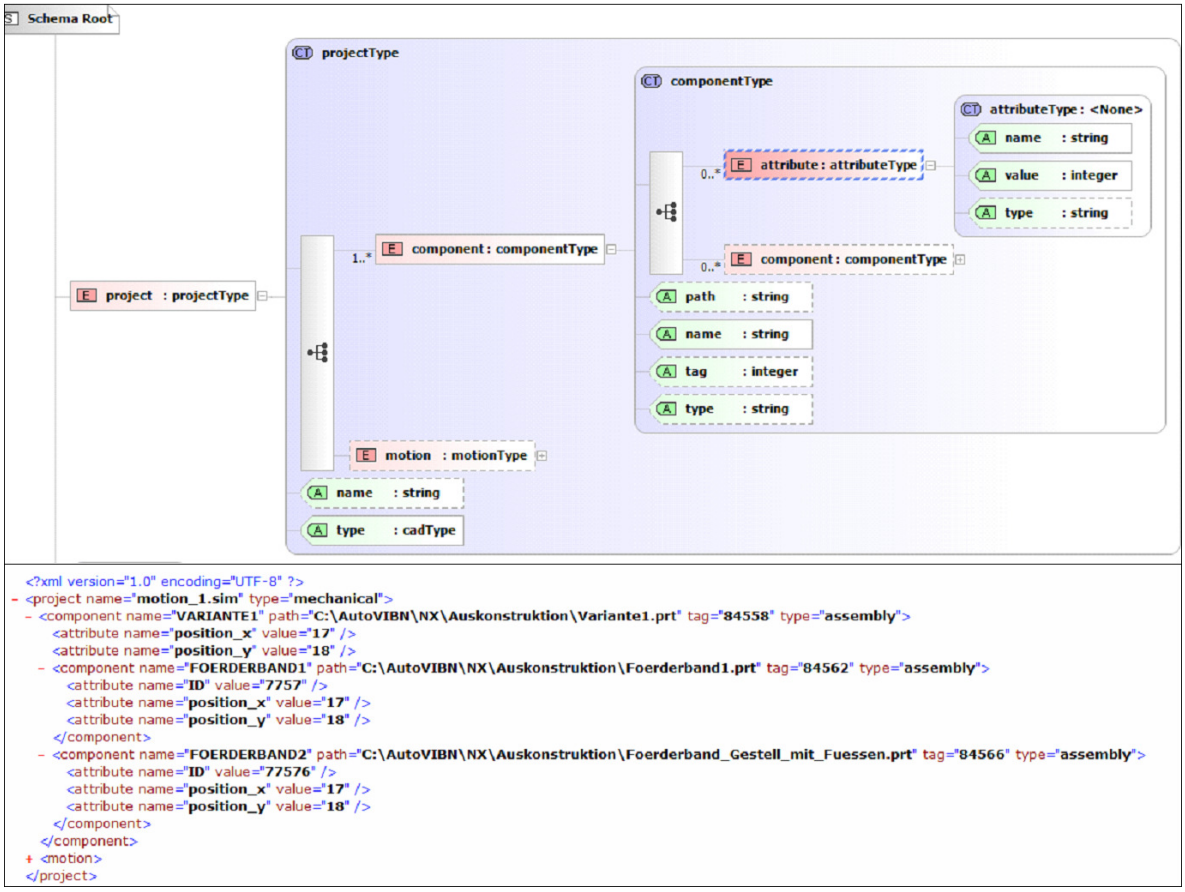



Abbildung 3.36: Datentransfer auf der Basis der XML

### 3.4.3 Extrahierte Daten

Im Rahmen des Projektes AutoVIBN kamen die CAD-Systeme NX der Firma Siemens PLM Software GmbH für die Mechanik und E3.series der Firma Zuken für die Elektrotechnik zum Einsatz. Letzteres kann auch für die Projektierung der Fluidik verwendet werden. Es wurden nachfolgend aufgelistete Daten extrahiert und als XML-Datei gespeichert:

MCAD

- Identifikationsnummer
- Positionen, Orientierungen von Bauteilen
- Form, Position und Orientierung von Detektoren (detectors)
- Bewegungsachsen (movers)
- Proprietäre Attribute der Bauteile

Die Geometrie der Bauteile wird in Form von VRML-Dateien (Virtual Reality Modeling Language) aus dem MCAD exportiert. Der Mechanismus zur Entfeinerung, Reduktion und Einbindung in die Funktionsbeschreibung ist in AP7 beschrieben.

ECAD

- Identifikationsnummer
- Betriebsmittel
- Verschaltung der Betriebsmittel
- Proprietäre Attribute der Betriebsmittel

### 3.4.4 Konsistenz zwischen Funktionsbeschreibung und CAD-Modellen

Von entscheidender Bedeutung für den Entwicklungsprozess ist die Konsistenz der Entwicklungsstände in den einzelnen Disziplinen. Ein Divergieren ist zu vermeiden, jedoch durch die fehlenden Hilfsmittel in der industriellen Praxis nicht immer zu verhindern. Dies zieht aufwändige Iterationsschleifen nach sich, die zu langen Entwicklungszeiten führen. Auch im Hinblick auf die Generierung von Simulationsmodellen sind konsistente Daten erforderlich.

In AP3 wurde der Mechanismus zur Konsistenzprüfung bereits ausführlich beschrieben. Die Voraussetzung zur Anwendung des vorgestellten Ansatzes ist die eindeutige Identifikation von Elementen in der Funktionsbeschreibung und den CAD-Systemen. Die Grundlage hierzu bilden die IDs. In der Funktionsbeschreibung werden diese automatisch vergeben. Für die CAD-Modelle wurden die systeminternen IDs genutzt, die zur Referenzierung von Modellbestandteilen dienen. Diese IDs werden automatisch der XML-Datei der Komponente beigefügt. Die Erstellung von Querbeziehungen erfolgt durch den Entwickler im Tracing-Modell. Hiermit lässt sich prüfen, ob Aktoren und Sensoren jeweils in den entsprechenden Modellkategorien existieren und übereinstimmen.

## 3.5 AP5/AP6: Automatische Generierung des Verhaltensmodells und Modellintegration in die Simulationsumgebung

Das Modell der Funktionsbeschreibung ist eine abstrakte Abbildung der zu projektierenden Maschine oder Anlage. Dieses Modell muss für eine VIBN um zusätzliche Informationen erweitert werden. Da gängige VIBN-Simulationswerkzeuge wesentliche Bestandteile des hier definierten Funktionsmodells nicht unterstützen, bspw. den Materialfluss oder die Kollisionserkennung, wurde ein Funktionsmuster eines Simulators erstellt, das den geforderten Ansprüchen gerecht wird.

### 3.5.1 Grundprinzip

Die Funktionsbeschreibung dient dazu, die Überführung der meist informell formulierten Kundenanforderungen in konkrete Konstruktionsmodelle zu unterstützen, siehe Abb. 3.37. Dieser Prozess wird in der industriellen Praxis meist durch text- und tabellenbasierte Notationsformen begleitet. Da diese weder formal noch ausführbar sind, ergibt sich eine starke Einschränkung hinsichtlich einer simulativen Analyse. Durch die hier vorgeschlagene Funktionsbeschreibung soll die Simulation von Beginn an mit eingebunden werden. Die Umsetzung unterschiedlicher Lösungsvarianten ist dadurch in einem frühen Stadium bewertbar. Es handelt sich bei der Funktionsbeschreibung jedoch noch immer um ein abstrahiertes Modell der Maschine, welches durch zusätzliche Informationen anzureichern ist, damit eine VIBN durchgeführt werden kann.

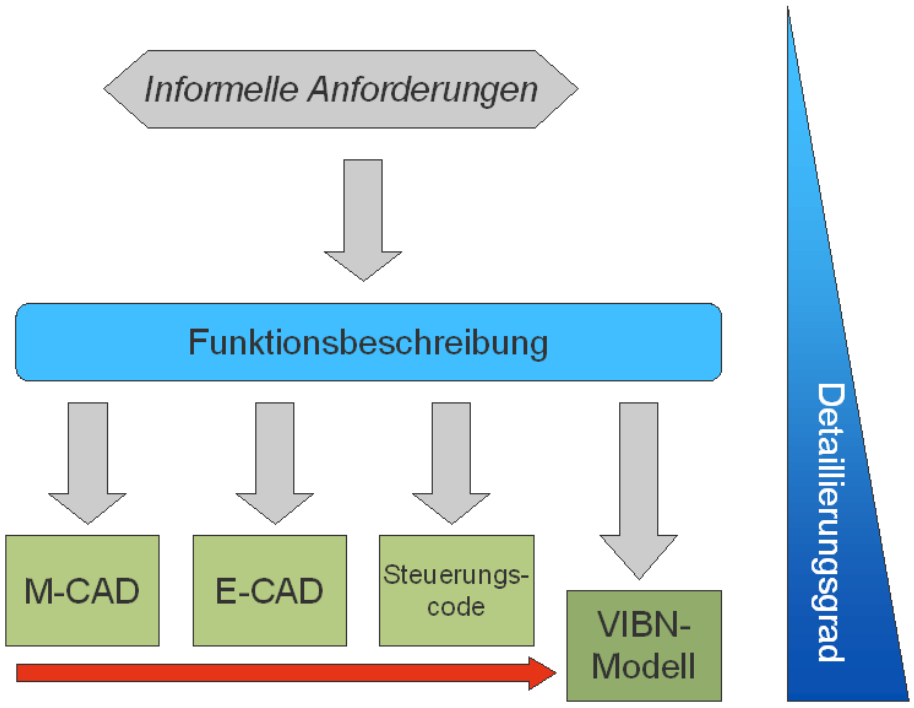



Abbildung 3.37: Einordnung der Funktionsbeschreibung in den Gesamtkontext

Das Grundprinzip des vorgeschlagenen Ansatzes besteht aus einer frühzeitigen Modellierung mit Hilfe der Funktionsbeschreibung, die dann mit Hilfe der CAD-Daten um zusätzliche Informationen angereichert wird, bevor die Ableitung des eigentlichen VIBN-Modells erfolgt, siehe Abbildung 3.38. Der Nutzen hierbei ist, dass die Funktionsbeschreibung wesentlich schneller und einfacher zu erstellen ist, als dies auf das VIBN-Modell zutrifft. Zudem kann das Funktionsmodell als Abstimmungs- und Kommunikationsmittel im Entwicklungsprozess herangezogen werden, was dessen Erstellung deutlich amortisiert.

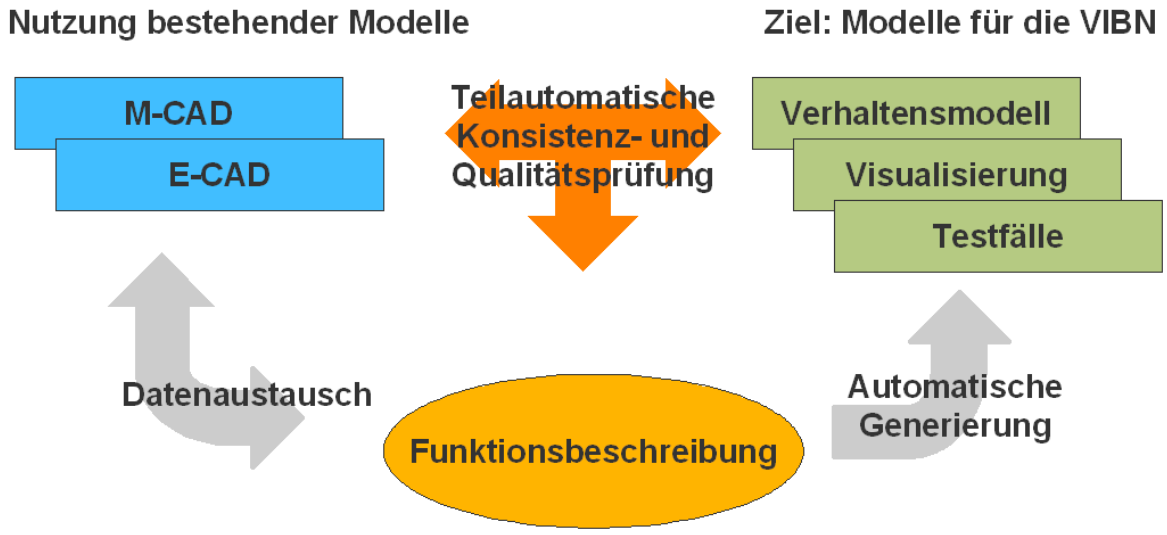



Abbildung 3.38: Funktionsbeschreibung als zentrales Element des Entwicklungsprozesses

### 3.5.2 Rückführung von Informationen aus den CAD-Systemen

Das Funktionsmodell stellt ein stark abstrahiertes Abbild der realen Maschine dar. In den CAD-Modellen sind allerdings Detailinformationen hinterlegt, die sich zur Erweiterung des Funktionsmodells nutzen lassen. Als Beispiel sei die Geometrie genannt, die zunächst mit Hilfe von Volumenprimitiven in der Funktionsbeschreibung hinterlegt werden kann. Eine derartige Darstellung ist für erste Analyseschritte ausreichend. Sollen jedoch Kollisionen oder die exakte Positionierung von Sensoren überprüft werden, so sind genauere Daten notwendig, wie sie im MCAD-Modell enthalten sind.

#### Einbindung von Daten aus dem MCAD

Aus dem Konstruktionsmodell der Mechanik können für die Funktionsbeschreibung vor allem die Geometriedaten herangezogen werden. Hierbei wurden konkret die Daten für die Parts, für die Detectors und für die Movers extrahiert. Im Folgenden wird für diese Modellelemente der automatisierte Import beschrieben.

Als Speicherform für 3D-Modelle hat sich in den letzten Jahren das VRML-Format etabliert. Da alle gängigen CAD-Systeme diese Art von Dateien unterstützen, wird für den hier entwickelten Ansatz ebenfalls darauf zurückgegriffen. Zwar können VRML-Modelle durch die CAD-Systeme sehr einfach ausgeleitet werden, jedoch müssen diese teilweise aufwändig nachbearbeitet werden. Dies ist dem Umstand geschuldet, dass bspw. die Bauteilhierarchie verloren geht oder sämtliche Bezeichnungen automatisiert umbenannt werden, was wiederum ein Auffinden von Elementen erschwert. Des Weiteren wird bei der Konstruktion der Maschine eine kinematisch korrekte Konstruktion vernachlässigt, was zur Folge hat, dass Bauteile einer Bewegungsgruppe nicht gemeinsam verfahren. Häufig werden in der industriellen Praxis andere Konstruktionsrichtlinien verfolgt, wie bspw. die Zusammenfassung von gemeinsam zu fertigenden Baugruppen, die aber im Hinblick auf die Simulation Anpassungen erfordern. Daher wird für diese Methode eine kinematisch korrekte Konstruktion gefordert, um automatisiert 3D-Daten in das Funktionsmodell übernehmen zu können.

Um dreidimensionale Bauteile aus dem MCAD-System in das Funktionsmodell übernehmen zu können, müssen zunächst die Beziehungen zwischen den beiden Modellkategorien erstellt werden. Hierzu dient das bereits in AP3 vorgestellte Modell für die Konsistenzprüfung (Tracing-Modell). Dieses wurde für den 3D-Import derart erweitert, dass nicht nur die Aktoren und Sensoren der unterschiedlichen Modelle miteinander verknüpft werden können, sondern auch eine direkte Zuweisung zwischen dem Funktionsmodell und der Mechanik möglich ist, siehe Abbildung 3.39. Mit Hilfe der IDs ist ein automatisches Auffinden der entsprechenden Bauteile im MCAD und eine Ausleitung von 3D-Daten möglich.

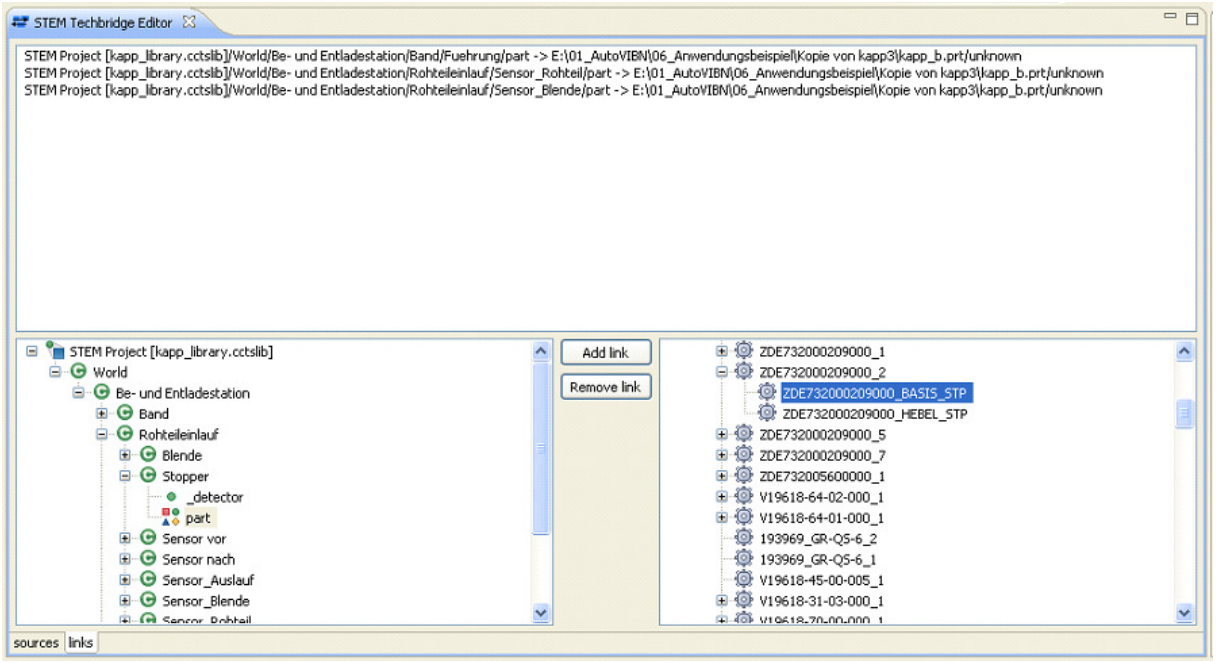



Abbildung 3.39: Verbindung zwischen dem Funktionsmodell und der Konstruktionsebene

Vor einem Export von Geometrieinformationen empfiehlt es sich jedoch, die für die Simulation nicht wesentlichen Teile zu entfernen, bspw. Schrauben oder Bohrungen. Hiermit kann

die Ausführung deutlich beschleunigt werden, da sich die Kollisionsrechnung deutlich vereinfacht. Daher werden zunächst alle als Standardbauteile gekennzeichneten Elemente automatisiert mit Hilfe der Programmierschnittstelle gelöscht. Dies umfasst auch Fasen und Kanten, die sich über die API ebenfalls identifizieren und entfernen lassen. Eine weitere, deutliche Verkleinerung der Dateigröße kann durch eine Polygonreduktion erzielt werden. Hierzu wurde das Open-Source-Werkzeug Meshlab eingesetzt, welches von CNR (Italian National Research Council) entwickelt wurde. Das zugehörige Programm lässt sich über den Meshlab-Server von extern ansteuern. Für die Reduktion an sich stehen unterschiedliche Verfahren zur Verfügung, wobei sich der Algorithmus *Clustering Decimation* als besonders geeignet erwiesen hat, da bei einer zufriedenstellenden Minimierung der Flächen- und Knotenanzahl nur wenige offene Strukturen entstehen.

Die reduzierten Bauteile werden mit der ID des Gegenparts in der Funktionsbeschreibung benannt, wobei hier auf die Informationen des Tracing-Modells zurückgegriffen werden kann. Die Zuweisung, die Positionierung und die Orientierung im Funktionsmodell finden dann automatisch, mit Hilfe der XML-Datei, statt.

Eine weitere wesentliche Verbesserung der Modellgüte lässt sich durch die Übernahme der Detektoren und Bewegungsachsen erzielen. Hierzu müssen diese jedoch explizit im MCAD-Modell angelegt werden. Dies hat den Vorteil, dass somit exakte Daten verwendet werden können, da in der Funktionsbeschreibung nur eine grobe Spezifikation möglich und auch in den frühen Entwicklungsphasen sinnvoll ist. Die entsprechenden Daten der Detektoren und Bewegungsachsen werden mit Hilfe der Programmierschnittelle des CAD-Systems im XML-Austauschformat hinterlegt. Dieser Schritt kann prinzipiell mit jedem beliebigen CAD-Werkzeug durchgeführt werden, vorausgesetzt, die implementierte Programmierschnittstelle ermöglicht den Zugriff auf die notwendigen Daten. Nach der Ausleitung der benötigten Informationen werden diese automatisiert analysiert und in Geometriedaten für das Funktionsmodell umgewandelt. Der Nutzen dieser Vorgehensweise besteht bspw. darin, dass die Positionen von Sensorbereichen wesentlich genauer in die Simulation mit eingebunden werden können. Da die Geometrie Bestandteil des Modells ist und nicht nur zur Visualisierung dient, kann somit überprüft werden, ob Objekte den Sensor aufgrund ihrer Verfahrbewegung auch wirklich auslösen und den Sensorbereich durchdringen. Mit Hilfe logischer Ersatzmodelle, wie sie üblicherweise für diese Zwecke eingesetzt werden, ist eine derartige Überprüfung nicht möglich. Im Hinblick auf die Bewegungsachsen lassen sich durch die Einbindung der CAD-Daten auch komplexe Verfahrbewegungen einfach darstellen.

#### Einbindung von Daten aus dem ECAD

Im ECAD-System sind Informationen über die verwendeten Aktoren und Sensoren hinterlegt. Sämtliche Betriebsmittel und deren Verschaltungen können aus dem ECAD entnommen werden. Die Beziehungen zwischen dem Funktionsmodell und dem ECAD-Modell sind, wie bereits für die Mechanik geschildert, im Tracing-Modell gespeichert. Das für die Versuchszwecke eingesetzte Werkzeug E3.series erlaubt zudem die Projektierung der Hydraulik und Pneumatik. Dadurch sind zusätzlich die Relationen zwischen der Elektrotechnik und der Fluidik abgebildet.

Da die Funktionsbeschreibung nur über einen begrenzten Informationsgehalt hinblicklich der technischen Daten auf der Konstruktionsebene verfügt, kann eine Erweiterung bzw. Verfeinerung des Maschinenverhaltens mit Hilfe der Daten aus dem E/FCAD erfolgen. Im Funktionsmodell sind die Aktoren und Sensoren vereinfacht abgebildet, da das funktionale Verhalten im Vordergrund steht. Die detaillierte Struktur der Steuerkette sowie die genaue Ausprägung des Arbeitsglieds können aus dem E/FCAD-Modell in das Funktionsmodell übernommen werden. Hierzu ist jedoch eine einmalige Abbildung des zu übernehmenden Elementes, bspw. ein Wegeventil oder ein Motor, in der Modellbibliothek erforderlich. Ebenso lassen sich technische Daten für die Sensorik gewinnen, die dann im Technischen Ressourcenmodell der zugehörigen Komponenten integriert werden. Dies umfasst z. B. die E/A-Belegung oder auch das Protokoll des Busteilnehmers, welches die Bedeutung der einzelnen Bitbereiche regelt. Auf das Technische Ressourcenmodell als Bindeglied zur Steuerungsebene wird in AP7 noch genauer eingegangen.

Die notwendigen Informationen werden aus dem ECAD-Werkzeug E3.series mit Hilfe der Programmierschnittstelle automatisch extrahiert und als XML-Datei abgelegt. Die darin enthaltenen Daten werden anschließend interpretiert und mit Hilfe der Querbeziehungen im Tracing-Modell den jeweiligen Komponenten im Funktionsmodell zugeordnet. Durch die beschriebene Vorgehensweise ist ein wesentlich realistischeres Verhalten darstellbar.

### 3.5.3 Analyse und Bewertung von Simulationssystemen

Im Rahmen des Forschungsvorhabens AutoVIBN wurden unterschiedliche Simulationswerkzeuge für die VIBN untersucht (Sinumerik MS, WinMOD, Process Simulate Commissioning). Dabei war das Ziel, ein geeignetes System für die Überführung des Funktionsmodells in ein VIBN-Modell zu identifizieren. Als Randbedingung wurde vorab formuliert, dass alle Modellbestandteile, wie z. B. das Geometriemodell, übernommen werden müssen. Eine Modifikation und Anpassung des Funktionsmodells sowie die redundante Haltung von Modellmodulen in der Zielumgebung zum Zwecke einer Kompositionsstrategie sollte vermieden werden. Der vorgestellte Ansatz ist generativer Art und unterscheidet sich hierdurch maßgeblich von Modularisierungs- und Standardisierungsstrategien, die eine Modellwiederverwendung anstreben.

Die am Markt verfügbaren Simulationswerkzeuge konnten jedoch die gestellten Anforderungen nicht erfüllen. Nachfolgend sind die wesentlichen Gründe hierfür zusammengefasst:

- Ungeeignete Schnittstellen zur Modellerstellung
- Geometriemodell und Kollisionserkennung
- Materialflusssimulation

Keines der untersuchten Simulationswerkzeuge im Bereich der VIBN unterstützt eine umfangreiche Modellerstellung von außen. Es werden lediglich Schnittstellen angeboten, die eine Verschaltung von bereits in der Entwicklungsumgebung bestehenden Modellblöcken erlaubt. Da hier das Funktionsmodell federführend ist und wie bereits erläutert keine Modellblöcke in externen Werkzeugen verwaltet werden sollen, um Brüche in der Werkzeugkette zu verhindern, hat sich dies als nachteilig herausgestellt. Von zentraler Bedeutung in der

Funktionsbeschreibung ist das Geometriemodell, da die Form eines Objektes Einfluss auf die Funktion hat. An dieser Stelle sind z. B. die zylindrisch modellierten Detektoren zu nennen, die als Lichtschranken fungieren. Eine Übernahme des Geometriemodells in das VIBN-Simulationsmodell gestattet ebenfalls keines der Werkzeuge. Dies trifft gleichermaßen auf den Materialfluss zu. Im Rahmen dieses Forschungsvorhabens wurde eine Methode erarbeitet, die es ermöglicht, den Materialfluss schnell und einfach abzubilden. Normalerweise werden hierfür meist aufwendig zu erstellende, logische Ersatzmodelle herangezogen.

Neben den bislang genannten Gründen existieren weitere Aspekte, wie z. B. unzureichende Kopplungsmöglichkeiten an die reale Steuerung, auf die aber nicht genauer eingegangen werden soll. Daher wurde für die VIBN ein Funktionsmuster eines Simulators entwickelt, das alle Anforderungen erfüllt. Im folgenden Abschnitt wird die Generierung des VIBN-Modells näher erläutert.

### 3.5.4 Generierung des VIBN-Modells

Um einen Informationsverlust zu verhindern wird das Funktionsmodell nicht in eine kommerzielle Simulationsumgebung transferiert, sondern als C++-Datei herausgeschrieben. Hierin sind sämtliche Kernelemente, über die auch das Funktionsmodell verfügt, enthalten. Es werden alle Komponenten einschließlich deren hybride Automaten, dem Geometriemodell und dem Materialfluss generiert. Der prinzipielle Aufbau des Simulators wird im nachfolgenden Absatz beschrieben.

#### Aufbau des Simulators

Es werden je nach Bedarf drei unterschiedliche Simulationsarten unterstützt. Dies sind die

- Vollsimulation,
- die Hardware-in-the-Loop-Simulation (HIL)
- und die Software-in-the-Loop-Simulation (SIL).

Für die Vollsimulation wird das Funktionsmodell einschließlich der modellierten Steuerung generiert. Mit Hilfe der Hardware-in-the-Loop-Simulation kann die reale Steuerung als Testobjekt mit eingebunden werden. Von Vorteil ist hier, dass auch die Steuerungshardware sowie die eingesetzten Feldbusverbindungen in den Test mit einbezogen werden können. Die Software-in-the-Loop-Simulation ist dadurch gekennzeichnet, dass das virtuelle Maschinenmodell an eine virtuelle Steuerung gekoppelt wird. Hierfür wurde ein virtueller Steuerungstyp der Firma Siemens verwendet, der in der Lage ist, zur IEC 61131 konformen Steuerungscode zu interpretieren.

Der ausgeleitete C++-Code ist nach Aufgaben, sog. Jobs, strukturiert. Diese sind im Folgenden aufgeführt und werden kurz beschrieben:

- AGJob: Generierung von neuem Material und Entfernung von altem Material in der Materialflusssimulation
- DelayJob: Stellt die minimale Zykluszeit sicher

- DetectorsJob: Berechnet die Kollisionen mit den Detektoren und deren Auslösung
- MotionJob: Bewegungskollisionen und Bewegungsberechnung
- SCSEPJob: Netzwerkkommunikation mit der Bedienung der Simulation und der Visualisierung
- SimulationJob: Ausführung der Simulation der Automaten

Des Weiteren werden die einzelnen Komponenten als C++-Programme generiert, worin z. B. die hybriden Automaten integriert sind. Für eine Vollsimulation sind die bisher vorgestellten Elemente ausreichend. Soll eine HIL- oder eine SIL-Simulation durchgeführt werden, so muss noch die jeweilige Kopplung zur realen oder virtuellen Steuerung generiert werden. Hierfür stehen wiederum entsprechende Jobs zur Verfügung:

- HIL-Simulation: SimbaJob
- SIL-Simulation: SILPlcSimJob

Für die Kollisionserkennung wurde, wie dies auch in der Funktionsbeschreibung implementiert ist, eine Open-Source-Bibliothek verwendet (Freesolid). Deren Funktionen können mit Hilfe einer sog. Dynamic Link Library (DLL) für die VIBN-Simulation genutzt werden.

### Technisches Ressourcenmodell

In der Funktionsbeschreibung werden für die Modellierung logische Signale genutzt. Diese stimmen nicht unbedingt mit den realen Steuerungsein- und -ausgängen überein. So kann ein Motor im Funktionsmodell z. B. lediglich über den Eingangsport *Run* verfügen, in der Realität aber über ein komplexes Steuerwort angesprochen werden. Die Übersetzung zwischen logischen und realen Signalen übernimmt das bereits erwähnte Technische Ressourcenmodell. Hierin können zum einen Zuweisungen zwischen den unterschiedlichen Signalkategorien getroffen werden, zum anderen lassen sich auch Bitbereiche innerhalb von Datenwörtern definieren, die wiederum Informationsblöcke, wie z. B. Beschleunigungsrampen, umfassen. Auf das Technische Ressourcenmodell wird hinsichtlich der Steuerungskopplung in AP7 genauer eingegangen.

Dieses Modell ist für die C++-Codegenerierung von großer Bedeutung, da für die HIL- und die SIL-Simulation die Umrechnung von logischen in reale Signale notwendig ist. Daher muss das Technische Ressourcenmodell mit der Beschreibung der Steuerungsein- und -ausgänge vor der Generierung definiert werden.

Eine weitere Aufgabe dieses Modells ist die Abstimmung und Koordination zwischen unterschiedlichen Fachabteilungen. Als gemeinsame Schnittmenge sind die Aktoren und Sensoren hinterlegt, für welche ein Klassifikationsschema entwickelt wurde, das sich an den Anforderungen des Maschinen- und Anlagenbaus orientiert.

## 3.6 AP7: Kopplung des Gesamtmodells an eine reale Steuerung und 3D-Visualisierung

### 3.6.1 Kopplung an die Steuerung

Eine der essentiellen Aufgaben des Technischen Ressourcenmodells wurde bereits in AP5/6 angedeutet. Für die Generierung des VIBN-Modells ist die Zuweisung von logischen und realen Signalen unerlässlich. Auf die genaue Umsetzung soll nun in diesem Abschnitt detaillierter eingegangen werden.

#### Erweiterung des Technischen Ressourcenmodells auf die Steuerungsebene

Das Prinzip der Kopplung von Funktionsmodell und Steuerung ist in Abbildung 3.40 dargestellt und sowohl für die HIL-Simulation als auch für die SIL-Simulation gültig. Die Fragestellung, die sich im Hinblick auf die Steuerungsanbindung ergibt, ist, wie die realen Steuerungsein- und -ausgänge mit den Ports im Funktionsmodell verknüpft werden können. Eine direkte Ein-zu-eins-Verbindung ist nicht immer möglich, da sich die Modellports aus logischen Verknüpfungen oder Umrechungen der realen Steuerungsein- und -ausgänge zusammensetzen. Teilweise werden auch nur Bereiche eines Datenwortes benötigt, als Beispiel seien hier komplexe Steuerwörter genannt, deren einzelne Bits eine unterschiedliche Bedeutung haben. Wie 3.40 zeigt, sind diese Bitbereiche auf die Modellports abzubilden. Hierfür dient das Technische Ressourcenmodell.

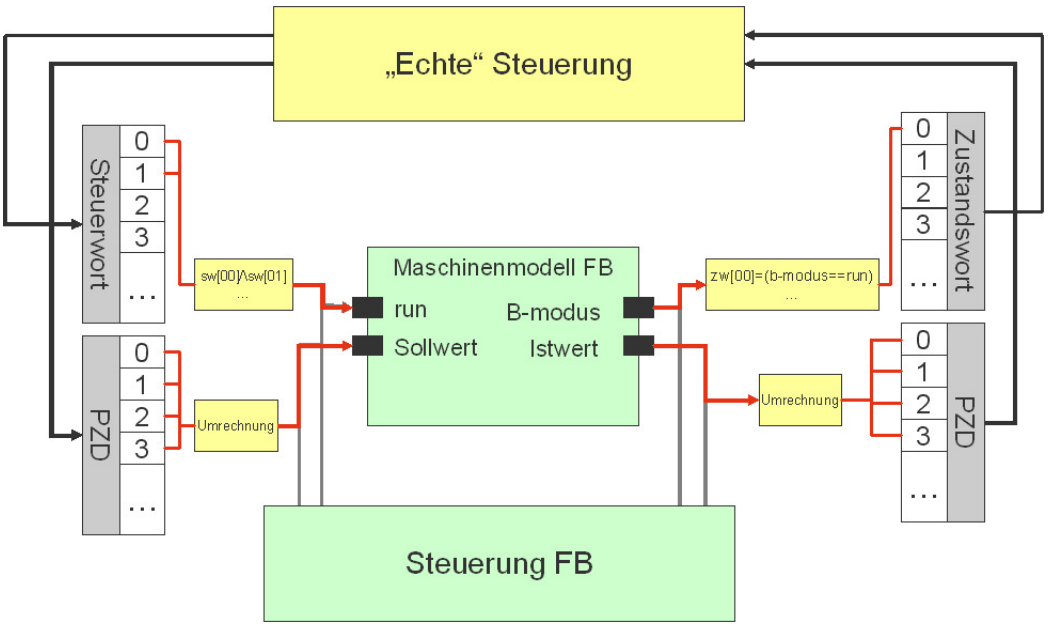



Abbildung 3.40: Technisches Ressourcenmodell als Bindeglied zwischen den Schnittstellensignalen des Funktionsmodells und der realen Steuerung

Zunächst müssen die Steuerungsein- und -ausgänge der projektierten Aktoren und Sensoren definiert werden, siehe Abbildung 3.41. Dies umfasst die physikalische Adresse auf der Steuerung und ebenso die Länge der zu empfangenden oder zu versendenden Daten.

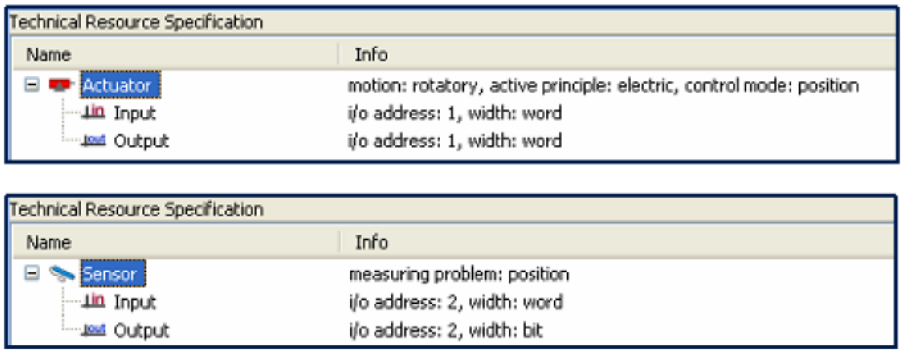



Abbildung 3.41: Definition der realen Steuerungsein- und -ausgänge im Technischen Ressourcenmodell

In einem nächsten Schritt sind die einzelnen Adressfelder, hier Packet Fields genannt, festzulegen. Diese haben die Aufgabe, einzelne Datenbereiche innerhalb eines festgelegten Adressbereiches zu definieren. Am Beispiel des Steuerwortes eines frequenzgeregelten Antriebs ist dies in Abbildung 3.42 verdeutlicht. Jedes der einzelnen Bits innerhalb des Steuerwortes hat eine spezielle Bedeutung, z. B. wird über Bit 11 die Drehrichtung invertiert.

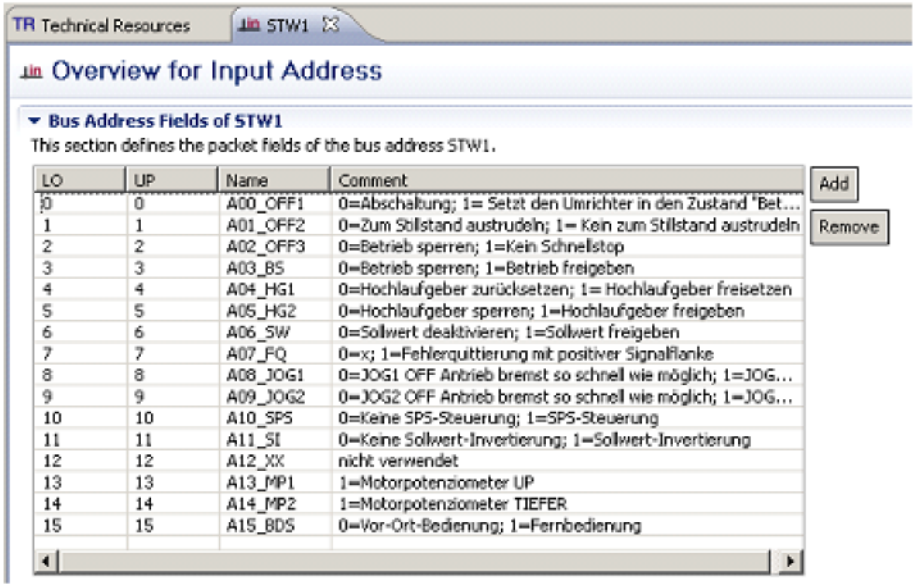



Abbildung 3.42: Adressfelder des Steuerwortes eines frequenzgeregelten Antriebs

Ein zweites Beispiel hierfür ist die Sollgeschwindigkeit desselbigen Antriebs, siehe Abbildung 3.43. Diese wird mit Hilfe eines Datenwortes beschrieben, wobei die ersten 15 Bit den eigentlichen Sollwert abbilden und das sechzehnte Bit das Vorzeichen festlegt.

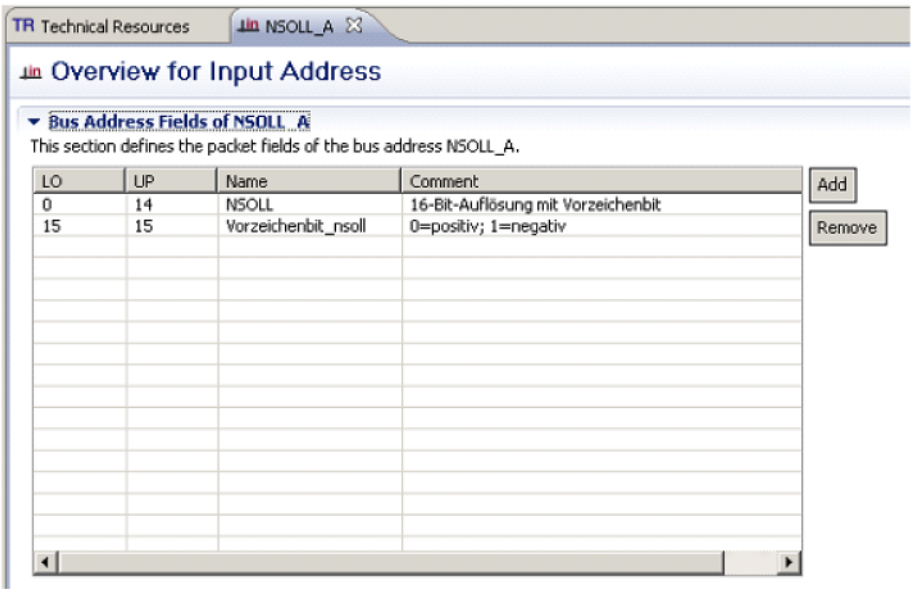



Abbildung 3.43: Adressfelder des Geschwindigkeitssollwertes eines frequenzgeregelten Antriebs

Die definierten Adressfelder müssen nun den einzelnen Ports des Funktionsmodells zugewiesen werden. Hierbei werden entweder logische Verknüpfungen auf der Basis boolescher Operatoren durchgeführt oder Umrechnungen vorgenommen. Diese Aufgabe übernehmen die sog. I/O-Adapter, wie sie exemplarisch in Abbildung 3.44 dargestellt sind.

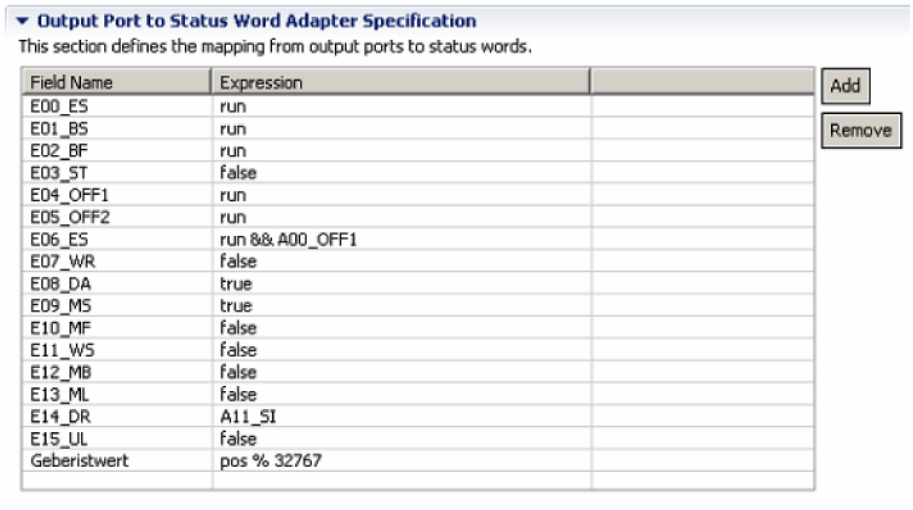



Abbildung 3.44: I/O-Adapter zur Verknüpfung von Adressfeldern mit den Ports des Funktionsmodells

### Hardware-in-the-Loop-Simulation

Die HIL-Simulation ist durch den Einsatz der realen Steuerung und eines virtuellen Maschinenmodells gekennzeichnet. Die automatische Generierung des VIBN-Modells wurde bereits in AP5/6 ausführlich erläutert. Im Folgenden wird nun erläutert, wie das VIBN-Modell an die Steuerung gekoppelt wird. Die Grundlage dazu bildet eine Simulationskarte der Firma Siemens, die sog. SimbaProPCI-Karte. Diese basiert auf FPGA-Technik und kann mit Hilfe eines PCI-Slots in den Simulationsrechner integriert werden. Die Karte verfügt je nach Ausführung über einen oder zwei Profibus-Anschlüsse. Vor der eigentlichen Simulation muss eine Projektierung erfolgen, d. h. sämtliche zu simulierende Profibus-Slaves müssen samt Adressen auf die Karte geladen werden. Dieser Schritt kann durch die Übernahme der Hardwarekonfiguration aus der Entwicklungsumgebung für die Steuerungssoftware (Simatic Manager) automatisiert werden. Hiernach ist eine fehlerfreie Kommunikation mit der realen Steuerung möglich.

Die Nutzdaten auf der Karte werden durch das entwickelte Funktionsmuster des Simulationssystems gelesen und geschrieben, siehe Abbildung 3.45. Auf der Simulationskarte wird die eigentliche Peripherie simuliert. Der Zugriff auf die Nutzdaten, bspw. Steuerwörter (STW) oder Prozessdatenwörter (PZD), erfolgt über die Programmierschnittstelle, die somit eine Kopplung mit der Simulationshardware erlaubt.

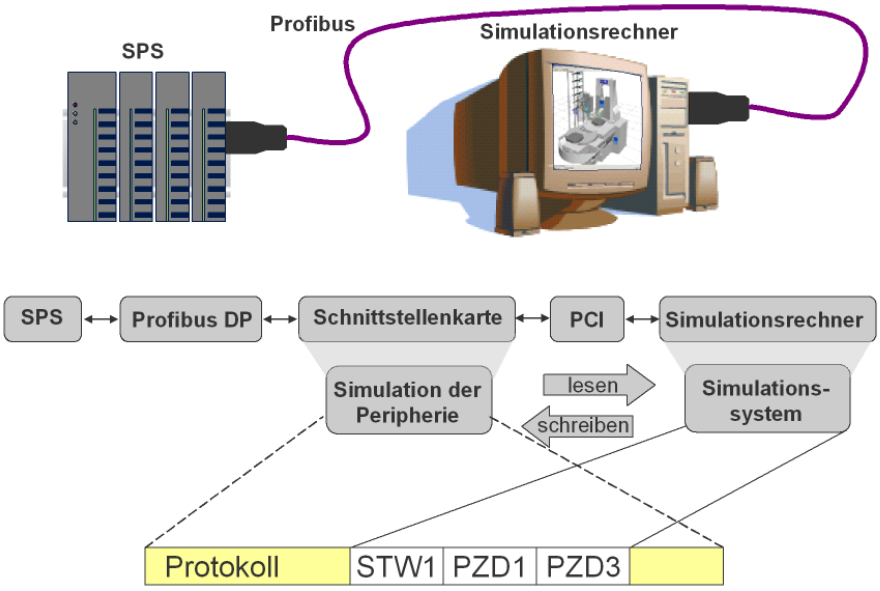



Abbildung 3.45: Aufbau der Hardware-in-the-Loop-Simulation

Die beschriebene Anbindung zur realen Steuerung wird automatisch bei der Generierung des C++-Codes mit herausgeschrieben, siehe Abbildung 3.46. Der sog. SimbaJob, siehe AP5/6, regelt die Kommunikation zur SPS.

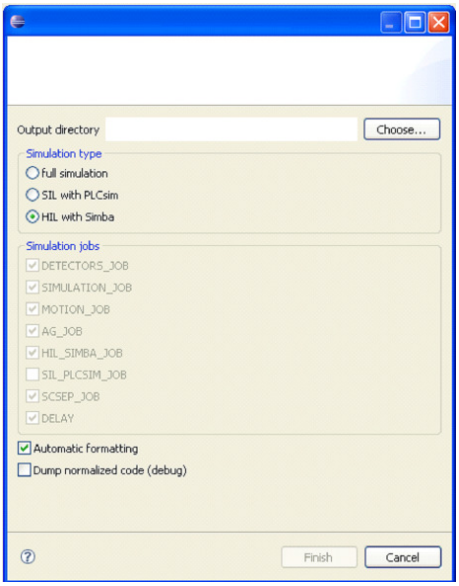



Abbildung 3.46: Generierung der HIL-Simulation

### Software-in-the-Loop-Simulation

Im Gegensatz zur HIL-Simulation wird bei der SIL-Simulation auch die Steuerung virtuell im Rechner abgebildet. Im Rahmen dieses Projektes wurde hierzu eine virtuelle Steuerung der Firma Siemens verwendet, die Teil der Entwicklungsumgebung Simatic Manager ist. Es handelt sich hierbei um das Werkzeug PLCSim. Das sog. PROSIM-Interface basiert auf der COM-Technologie (Component Object Model) und erlaubt einen Zugriff auf die simulierten Steuerungsein- und -ausgänge, siehe Abbildung 3.47. Der Takt der virtuellen Steuerung wird mit dem des Simulators gekoppelt. Eine Weiterschaltung erfolgt erst, wenn das VIBN-Modell berechnet wurde.

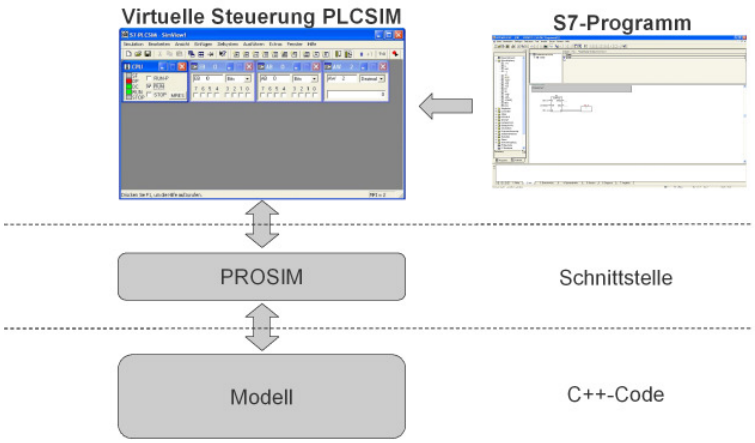



Abbildung 3.47: Prinzipieller Aufbau der SIL-Simulation

Wie es bereits für die HIL-Simulation erläutert wurde, erfolgt auch für die SIL-Simulation eine Generierung der Steuerungskopplung. Diese ist im sog. PLCSimSILJob implementiert.

### 3.6.2 Visualisierung

Die Visualisierung eines VIBN-Modells ist von besonderer Bedeutung, um Bewegungen der Maschine nachverfolgen zu können. Des Weiteren wird in diesem Absatz ein Unterpunkt der Bedienung des Systems gewidmet, da die Kopplungsmechanismen für diese und die Visualisierung auf die gleiche Art und Weise umgesetzt sind.

#### Generierung des 3D-Modells

Die Erstellung des 3D-Modells wurde schon in AP5/6 geschildert. Hierzu wurden VRML-Daten aus dem MCAD verwendet. Mit Hilfe dieser werden auch die Kollisionsrechnungen durchgeführt. Die Visualisierung ist hiervon unabhängig und dient nur zur Darstellung von Bewegungen. Dies hat den Vorteil, dass diese nicht zwangsläufig auf dem Simulationsrechner ablaufen muss. Durch die Verwendung eines zweiten Rechners, der rein zu Visualisierungsaufgaben herangezogen wird, kann der Simulationscomputer entlastet werden. Zudem sind hierdurch größere Zykluszeiten für die Visualisierung möglich. Die Kopplung des Maschinenmodells und der Visualisierung erfolgt über die Protokollfamilie TCP/IP. Das abgeleitete 3D-Modell ist in Abbildung 3.48 dargestellt.

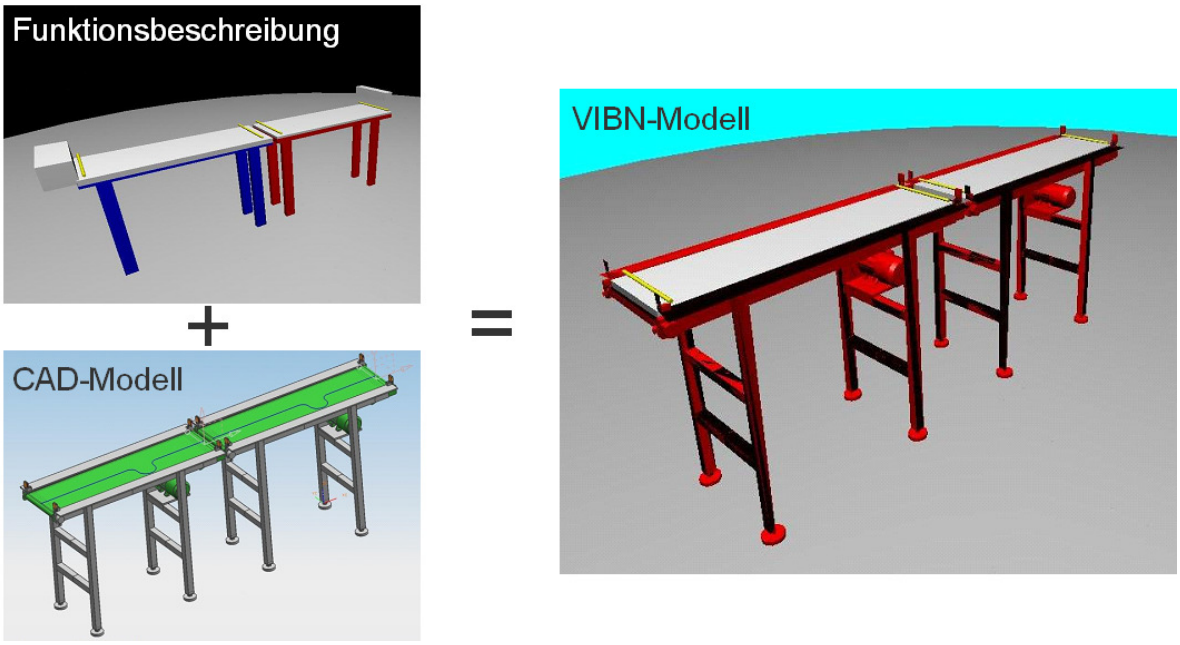



Abbildung 3.48: Erweiterung des Funktionsmodells mit CAD-Daten

Als Werkzeug zur Visualisierung wird das Softwarewerkzeug der Funktionsbeschreibung genutzt. Dadurch beschränkt sich der Darstellungsbereich nicht nur auf die reine 3D-Geometrie,

sondern erlaubt auch die Visualisierung weiterführender Aspekte. Hier sind vor allem die Automatenmodelle zu nennen, womit aktuelle Zustände und Variablen verfolgt werden können, siehe Abbildung 3.49.

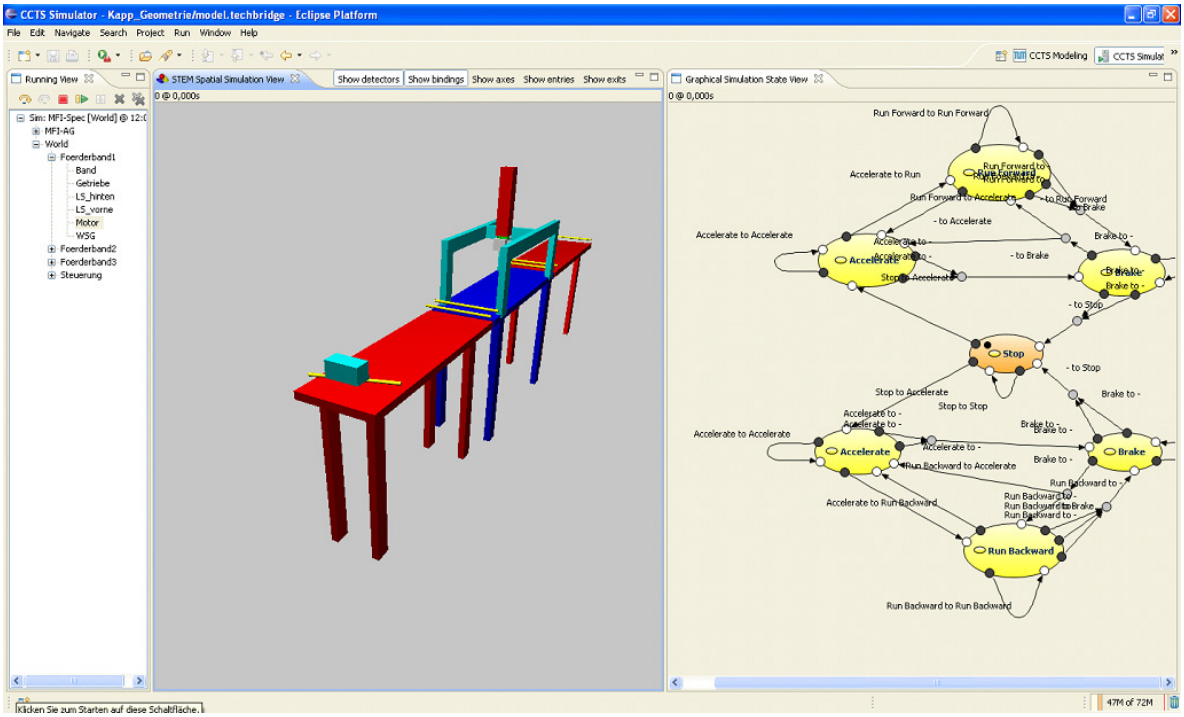



Abbildung 3.49: Darstellung simulationsrelevanter Aspekte am Beispiel des Automatenmodells

Zusätzlich sind Signaldiagramme im beschrieben Funktionsmuster hinterlegt, mit welchen sich die zeitlichen Verläufe verdeutlichen lassen, siehe Abbildung 3.50.

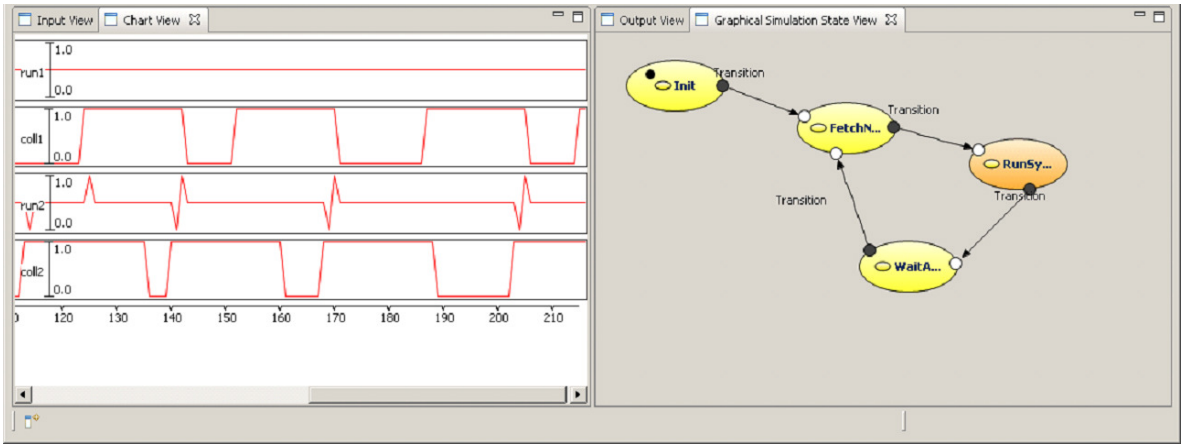



Abbildung 3.50: Darstellung von Signalverläufen während der Simulation

### Generierung von Bedienbildern

Neben der Visualisierung ist auch eine Schnittstelle zwischen Simulation und Bediener notwendig. Dies ist z. B. erforderlich wenn Material manuell in das System eingebracht oder Stellelemente durch das Bedienpersonal der Anlage betätigt werden sollen. Die Bedienbilder werden zunächst im Funktionsmodell projektiert. Hierfür stehen übliche Elemente, wie sie auch aus den gängigen Simulationswerkzeugen bekannt sind, zur Verfügung. In AP2 wurden die Bedienbilder als Teil einer Komponente vorgestellt. Der hierarchische Aufbau ermöglicht zudem die Erstellung komplexer Bediensysteme. Die Kommunikation mit anderen Komponenten erfolgt über die Ports. In Abbildung 3.51 wird ein Beispiel für ein Bedienbild aufgeführt.

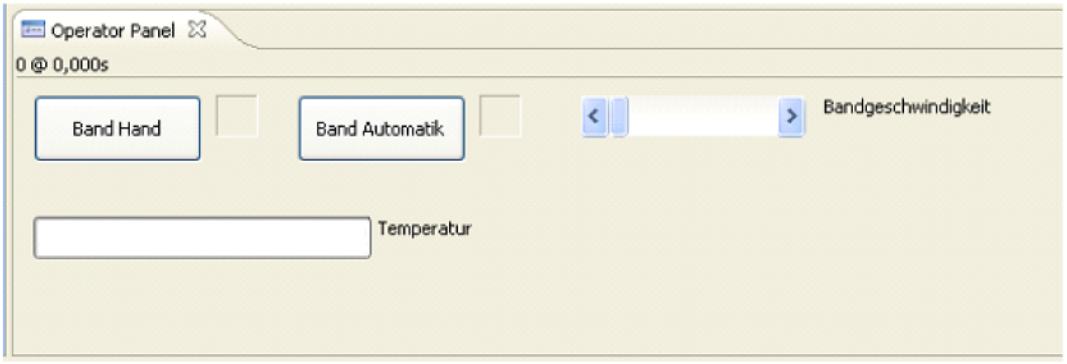



Abbildung 3.51: Elemente zur Steuerung der Simulation

Da die Bedienelemente auch für das VIBN-Modell benötigt werden, sind diese bei der Generierung zu berücksichtigen. Die technische Lösung entspricht der Visualisierung. Die Kopplung ist ebenfalls über das Netzwerkprotokoll TCP/IP realisiert.

## 3.7 AP8: Konkretisierung der Qualitätsmerkmale und Testfallerzeugung

In diesem Arbeitspaket lag der Fokus auf der Abbildung von unerwünschtem Verhalten und der automatischen Erzeugung von Testfällen auf der Modellebene. Beides wird nachfolgend näher erläutert. Zuvor soll aber ein kurzer Überblick über das Testen selbst gegeben werden.

### 3.7.1 Testen und modellbasiertes Testen

Beim Testen soll geprüft werden, ob ein System sich an funktionale Anforderungen hält. Dies erfolgt durch gezielte Stimulation und Auswertung der Systemreaktion. Das zu testende System wird meist als *System-under-Test* (SUT) bezeichnet. Der Testablauf ist in Abbildung 3.52 dargestellt. Das SUT selbst wird auf der Basis von Anforderungen konstruiert. Woher die *Testdaten* und das *erwartete Verhalten* kommen, ist beim klassischen Testen nicht näher spezifiziert. Beide zusammen werden auch als *Testfall* bezeichnet. Die Testdaten können für reine Softwaresysteme aus Systemeingaben bestehen, im Allgemeinen beinhalten diese jedoch eine Folge von Arbeitsschritten, die am System durchgeführt werden. Das erwartete Verhalten ist nicht notwendigerweise eine einzelne Systemreaktion, sondern kann auch aus einer Menge von zulässigen Systemantworten bestehen. Dies ist insbesondere für nichtdeterministische Systeme relevant oder im Falle einer Unterspezifikation. So könnte bspw. eine bestimmte Reaktion nach spätestens 30 Sekunden erwartet werden. Tritt das Ereignis schon nach 29 Sekunden auf, wird der Test häufig aber auch als erfüllt angesehen.

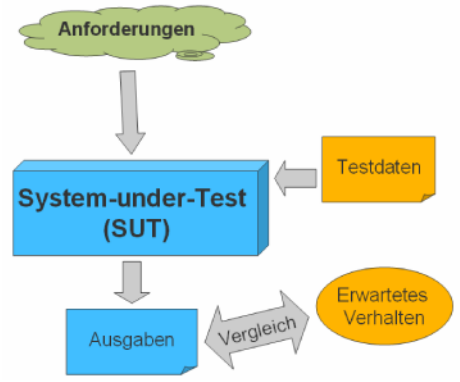



Abbildung 3.52: Artefakte beim klassischen Testen

Eine mögliche Antwort für die Herkunft der Testfälle liefert das modellbasierte Testen (vgl. Abbildung 3.53). Hierbei wird auf der Basis der Anforderungen nicht nur das SUT selbst konstruiert, sondern auch ein *Testmodell* beschrieben. Das Modell dient zum einen als Quelle für Testeingaben, die mit Hilfe entsprechender Verfahren generiert werden. Zum anderen liefert es auch die Menge des erwarteten bzw. des erlaubten Systemverhaltens.

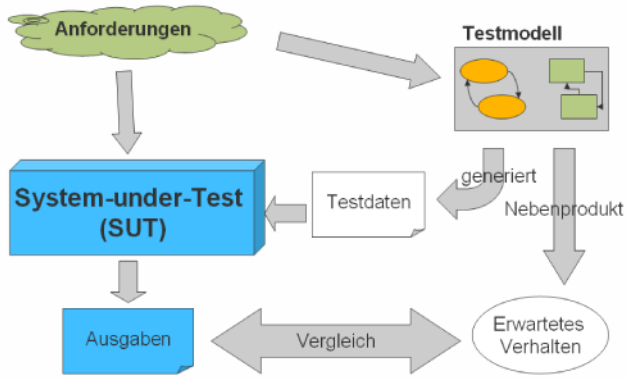



Abbildung 3.53: Artefakte beim modellbasierten Testen

Ein wichtiges Problem beim modellbasierten Testen ist die Abstraktionslücke zwischen dem realen System (SUT) und dem abstrakten Testmodell. Da die Generierung von Testfällen auf dem abstrakten Testmodell erfolgt, sind diese meist nicht direkt auf das SUT anwendbar. Ebenso ist das Verhalten des SUT konkreter als das des Testmodells. Die Überbrückung erfolgt durch einen sog. Testtreiber, wie in Abbildung 3.54 gezeigt. Dieser ist für die Übersetzung zwischen den konkreten und den abstrakten Signalen und Reaktionen zuständig. Da dieses Problem genauso während der VIBN bei der Kopplung zwischen dem konkreten Steuerungsprogramm und dem abstrakten Maschinenmodell auftritt, lässt sich die in den Arbeitspaketen 5 bis 7 erarbeitete Vorgehensweise auch hier anwenden.

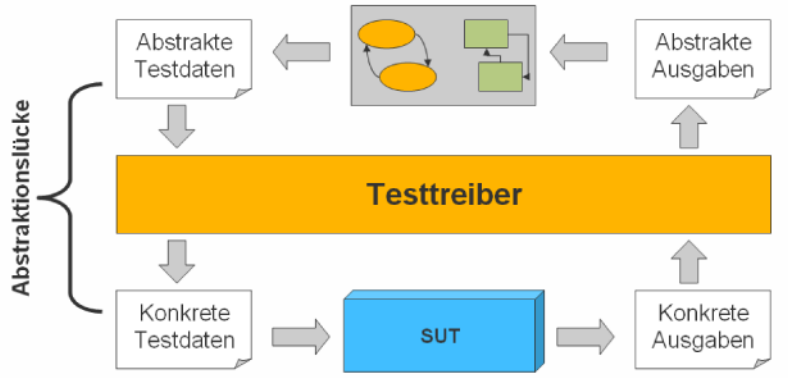



Abbildung 3.54: Problem der Abstraktionslücke bei der Generierung von Testfällen

### 3.7.2 Modellierung von unerwünschtem Verhalten durch Zusicherungen

Wie zuvor beschrieben soll sich aus dem Testmodell das erwünschte oder zulässige Verhalten ableiten lassen. Das Ziel hierbei ist, die Funktionsbeschreibung (also das Verhaltensmodell) direkt als Testmodell zu nutzen. Als Mittel zur Definition des zulässigen Verhaltens werden Zusicherungen eingesetzt, deren Verletzung zu einem unerwünschten Zustand führen.

Bei der Einführung der Zusicherungen war die Intention, möglichst wenig neue Modellelemente zu ergänzen. Als Lösungsweg wurde daher die Verwendung verbotener Zustände gewählt. Diese Zustände werden im Modell markiert und in roter Farbe dargestellt. Wird während der Simulation ein solcher Zustand erreicht, so kommt dies einer Zusicherungsverletzung gleich und ein Abbruch erfolgt. Ein Beispiel ist in Abbildung 3.55 verdeutlicht. Dieser Automat stellt einen Positionsregler dar, der im Zustand *Main* operiert. Falls eine Bewegungsvorgabe des Automaten nicht umgesetzt werden kann, wird die Transition zum Zustand *Blocked* aktiviert. Da dieser als verboten markiert ist, wird also das Blockieren des Antriebs als unerwünschtes Fehlverhalten interpretiert.

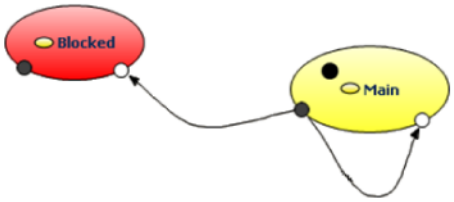



Abbildung 3.55: Automat mit verbotenem Zustand

Unter Verwendung von Detectors lassen sich auch geometrische Zusicherungen formulieren. Das in Abbildung 3.56 skizzierte Beispiel zeigt ein Modell einer Fertigungsstraße zur Reifenmontage. Eine wichtige Zusicherung ist, dass während der Montage keine Reifen zu Boden fallen dürfen. Dies wird im Modell so umgesetzt, dass ein großer Detector (in der Abbildung in gelber Farbe) die Fläche des Bodens darstellt. Der zugehörige Automat schaltet bei der Kollision eines beliebigen Parts mit dem Boden auf den verbotenen Zustand *Collided* um.

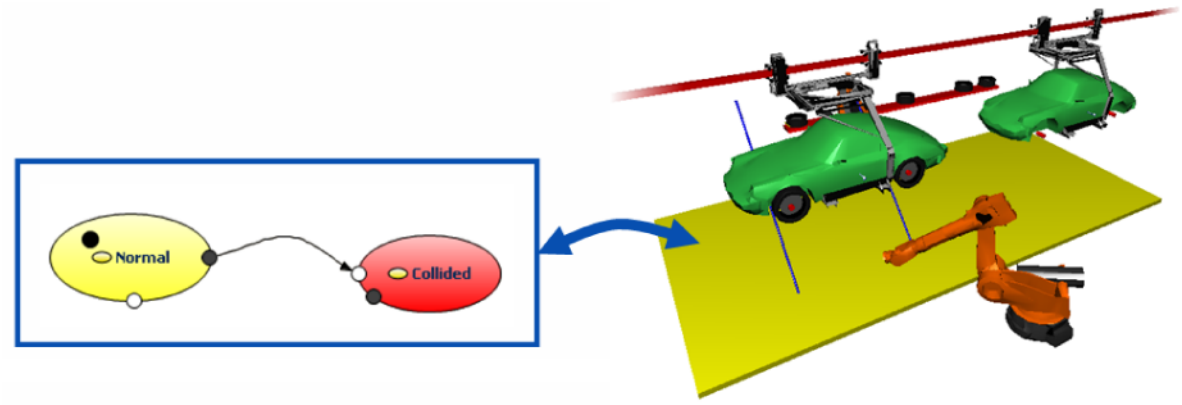



Abbildung 3.56: Modellierung einer Zusicherung, die das Herabfallen von Gegenständen verbietet

Insgesamt bietet die Modellierung der Zusicherungen durch verbotene Zustände den wesentlichen Vorteil, dass die Modellierungssprache nur minimal erweitert werden muss. Dadurch reduziert sich für einen Anwender der nötige Lernaufwand, aber auch die verarbeitenden Werkzeuge können einfach gehalten werden. Andere Beschreibungstechniken, die

häufig zur Formulierung von Zusicherungen verwendet werden, wie etwa Ausdrücke in temporaler Logik (LTL, CTL), lassen sich verlustfrei in Automaten umwandeln. Die Ausdrucksstärke der Zusicherungen leidet also unter diesem einfachen Modell nicht.

### 3.7.3 Generierung von Testfällen

Die automatische Generierung ist ein Thema, das insbesondere im Software-Engineering im Kontext der modellbasierten Entwicklung intensiv untersucht wurde. Ein Ziel dieses Arbeitspakets war die Untersuchung, ob und wie sich diese Verfahren auch auf einen modellbasierten Entwicklungsprozess für mechatronische Systeme übertragen lassen.

#### Existierende Verfahren

Für Softwaresysteme wurde in den vergangenen Jahrzehnten eine Vielzahl unterschiedlicher Methoden zur Testfallgenerierung vorgeschlagen. Bei genauer Betrachtung handelt sich eigentlich um eine Generierung der Test*eingaben*, während der Testfall auch das erwartete Verhalten umfasst. Im Sprachgebrauch hat sich aber der Begriff der Testfallgenerierung durchgesetzt. Die Ansätze lassen sich in verschiedene Klassen einteilen, die nachfolgend kurz vorgestellt werden.

Die Eingabe für alle diese Verfahren ist ein abstraktes Modell des Systems. Üblicherweise geht man davon aus, dass dieses Modell als Zustandsmaschine dargestellt ist, an deren Transitionen komplexe, boolesche Ausdrücke als *Guards*, d. h. Bedingungen, die erfüllt sein müssen, um dieser Transition zu folgen, stehen können.

**Generierung nach Überdeckungskriterien** Bei der Generierung nach Überdeckungskriterien ist das Ziel eine Menge von Testeingaben zu finden, die das Testmodell bezüglich eines bestimmten Kriteriums überdecken. Der Hintergrund hierbei ist, dass eine starke Überdeckung angestrebt wird, da nicht überdeckte Teile auch von keinem Test erfasst werden. Zudem schreiben einige Entwicklungsrichtlinien vor, bspw. im Luftfahrtbereich, dass die benutzte Testmenge eine bestimmte Überdeckung erreichen muss. Oft soll bei einer Generierung gleichzeitig die Zahl der Testfälle minimiert werden, da jeder Testfall in der Durchführung wieder Zeit und Ressourcen bindet.

Typische Überdeckungskriterien sind die Zustandsüberdeckung, bei der jeder Zustand bei Ausführung aller Testfälle mindestens einmal durchlaufen werden muss, und die Transitionsüberdeckung, bei der jede Transition ausgeführt werden soll. Man kann auch die Transitionsbedingungen in das Überdeckungskriterium einbeziehen, wenn sich diese aus mehreren Teilausdrücken (etwa Konjunktionen) zusammensetzen. So fordert die Bedingungsüberdeckung, dass jede Teilbedingung einmal zu *wahr* und einmal zu *falsch* ausgewertet wird. Da dies in der Praxis oft nicht erreicht werden kann, weicht man häufig auf die *modified condition/decision coverage* (MC/DC) zurück, die hier weniger restriktive Anforderungen stellt.

Für die eigentliche Generierung der Testfälle gibt es mehrere Verfahren. Allen ist gemein, dass sie sowohl das Testmodell (den Zustandsgraphen) und das gewählte Überdeckungskriterium in einen mathematischen Formalismus übersetzen, um dann einen Lösungsalgorithmus einzusetzen. Die gefundene Lösung lässt sich üblicherweise als eine Testeingabe interpretieren und rückübersetzen. Das mathematische Modell wird dann angepasst und der Prozess wiederholt sich so lange, bis die gewünschte Überdeckung erreicht ist. Die Herausforderung liegt hierbei meist in der Wahl eines geeigneten Formalismus und dann in der Übersetzung von Modell und Kriterium in diesen.

Die am häufigsten benutzten, mathematischen Formalismen sind *Constrain Logic Programming* (CLP), *SAT-modulo-Theories* (SMT) und Varianten von *Model Checking*. Alle diese Ansätze sind auch in anderen Bereichen weit verbreitet, was den Vorteil bietet, dass die Lösungsverfahren sehr ausgeprägt entwickelt sind. Allerdings gestalten sich die zu lösenden Probleme in diesen Formalismen (wie auch das zugrundeliegende Problem der Testfallgenerierung) sehr komplex und rechenintensiv. Formal betrachtet sind diese Probleme (je nach Kodierung der Eingaben) mindestens NP-hart oder auch PSPACE-hart, was in der Praxis bedeutet, dass nur Lösungsverfahren mit exponentieller Laufzeit existieren können. Dennoch lassen sich mit entsprechenden Heuristiken oft auch große Instanzen lösen und entsprechend Testeingaben für sehr umfangreiche Systeme erzeugen.

Obwohl die Steuerung der Testfallerzeugung mittels Überdeckungskriterien sehr elegant erscheint, da keine weiteren Angaben nötig sind, ist die Qualität und die Aussagekraft der damit erreichbaren Testeingaben sehr umstritten. Während die generierten Testfälle natürlich das Modell überdecken, haben Experimente gezeigt, dass viele Fehler, die durch von Hand erstellte Tests gefunden werden, von diesen Verfahren nicht identifiziert werden. Gleichzeitig lokalisieren die generierten Tests aber auch Fehler, die von den handgeschriebenen Varianten nicht erkannt wurden. Somit scheint das Generieren von Testfällen nach Überdeckungskriterien eine gute Ergänzung zur manuellen Testerstellung zu sein, aber kein Ersatz.

**Generierung nach expliziten Testzielen** Das Verfahren ist ähnlich zu den Überdeckungskriterien, jedoch wird hier die Erstellung eines Testfalls durch ein Testziel gesteuert, das ein Testingenieur explizit vorgibt. Ein solches Testziel kann bspw. die Erreichung mehrerer Zustände oder Transitionen in einer gegebenen Reihenfolge umfassen oder auch Zustände die dabei nicht verwendet werden sollen. Die eigentliche Generierung der Testeingaben erfolgt dann auf die gleiche Art und Weise, wie bei den vorgestellten Verfahren der Transformation in einen mathematischen Formalismus, nur dass hier nicht das Überdeckungskriterium, sondern das Testziel transformiert wird.

Durch die Einbindung von manuell erstellten Testzielen sind diese Testfälle näher an den händisch definierten Ausprägungen positioniert. Trotzdem wird der Testingenieur entlastet, da nicht die komplette Testeingabe, sondern nur das grobe Ziel vorgegeben werden muss. Es bleibt jedoch der Nachteil der umfangreichen Vorarbeiten durch den Anwender und auch das Problem der Laufzeit hinsichtlich der Berechnungsverfahren besteht wie im vorherigen Fall.

**Random-Testing** Beim *Random-Testing* werden die Eingaben vollständig zufällig erzeugt. Durch die Wahl der Wahrscheinlichkeitsverteilung lassen sich diese zu einem gewissen Grad beeinflussen. Der Testfall entspricht somit einem *random walk* auf dem Testmodell. Neuere Verfahren nutzen auch die Ergebnisse aus früheren Läufen, um die Qualität der Testeingaben zu verbessern.

Ein wesentlicher Vorteil des Random-Testing ist der vergleichsweise geringe Aufwand, der zur Erzeugung der Testfälle benötigt wird. Auch der Berechnungsaufwand ist im Vergleich zu den beiden vorher genannten Verfahren niedrig. Allerdings wird die Güte und die Aussagekraft der erzeugten Testfälle noch immer intensiv diskutiert. Aktuelle, experimentelle Ergebnisse deuten darauf hin, dass in der Praxis nicht die Qualität der von Hand geschriebenen Testfälle erreicht wird. Dennoch können die generierten Tests sehr viele Fehler finden, insbesondere auch solche, die die von Experten definierten Tests nicht lokalisieren konnten.

### 3.7.4 Praktische Umsetzung und Erprobung

Initial wurden Versuche unternommen, die Funktionsbeschreibung in einen für die Testfallerzeugung geeigneten Formalismus zu überführen oder eine Testfallgenerierung mit Hilfe einer expliziten Zustandsraumtraversierung zu realisieren. Dabei zeigte sich, dass die Abbildung komplexer, geometrischer Transformationen und der Kollisionsrechnung in den gängigen Formalismen nicht bzw. nur sehr umständlich möglich ist. Diese schlechte Formulierbarkeit führt zusammen mit der ohnehin schon vergleichsweise hohen Modellgröße zu extrem großen, mathematischen Modellen, die sich nicht innerhalb einer überschaubaren Zeit (d. h. mehrerer Tage) lösen lassen. Ebenso scheitert die explizite Traversierung des Modells am sehr großen Zustandsraum, der durch die geometrischen Positionen bedingt ist. Dieser führt zusammen mit den zahlreichen Quellen für den Nichtdeterminismus zu einer *Zustandsexplosion*.

Es soll nicht ausgeschlossen werden, dass sich die eben genannten Probleme durch geeignete, mathematische Ansätze reduzieren lassen. Allerdings ist zu erwarten, dass hierfür einige Jahre an Forschungsaufwand nötig wären. Um jedoch erste Erfahrungen mit den entsprechenden Techniken zu sammeln, wurde die Umsetzung des Random-Testing für die Funktionsbeschreibung beschlossen.

Für die Generierung von Testfällen ist wichtig, welche Eingänge das System besitzt, d. h. an welchen Stellen überhaupt Testdaten eingespeist werden können. Am offensichtlichsten sind hier die Bedienbilder, da alle Schalter und Eingabefelder natürlich auch Eingaben für Tests liefern. Eine weitere Quelle stellt der Nichtdeterminismus innerhalb des Systems dar, etwa wenn in einem Automat mehrere Transitionen bereit sind. Die Wahl der dann benutzten Transition kann als Eingabe interpretiert werden. Das gilt insbesondere für das an den Entries einlaufende Material. Die genaue Position und der Zeitpunkt der Erzeugung ist meist nichtdeterministisch gewählt und wird somit als Eingabe gesehen. Da die Beschaffenheit des eingehenden Materialflusses (wie z. B. der Abstand zwischen den Teilen) das Verhalten der Maschine wesentlich beeinflusst, ist die Berücksichtigung beim Erstellen von Testeingaben von Bedeutung.

Die zweite Frage, die sich stellt, ist, wie die Eingaben erzeugt werden. In ersten Versuchen hat sich gezeigt, dass eine gleichverteilte Ziehung der Werte in jedem Simulationstakt fast

immer zu Eingaben führt, die einem zufälligen Rauschen entsprechen. Ein Knopf wird bei einem Simulationstakt von 50 ms damit im Erwartungswert zehnmal betätigt, was nicht einer typischen Benutzung entspricht. Genauso wird erzeugtes Material den nahezu gleichen Abstand haben und höchstens um wenige Takte Zeitabstand versetzt sein. Bessere Ergebnisse lassen sich durch ein zweistufiges Verfahren erreichen, bei dem für jeden Eingangswert erst mit einer Änderungswahrscheinlichkeit $\rho$ entschieden wird, ob ein neuer Wert generiert werden soll oder die Eingabe vom vorherigen Takt zu benutzen ist. Nur im Falle einer Änderung wird dann ein neuer Wert zufällig und gleichverteilt erzeugt. Dadurch wird erreicht, dass die Änderungsrate einer Exponentialverteilung entspricht und die Eingaben lange Phasen mit gleichen Werten enthalten. Die Länge dieser Phasen hängt von der Wahl des Parameters $\rho$ ab.

## 3.8 AP9: Umsetzung der Methode anhand eines Funktionsmusters

### 3.8.1 Entwicklungsprozess

Im Rahmen des Projektes AutoVIBN wurde eine neue Modellierungstechnik entwickelt, die es ermöglicht, Maschinen und Anlagen bereits in frühren Entwicklungsphasen zu modellieren. Da jedoch neben der Beschreibungstechnik auch die Art und Weise, wie diese eingesetzt wird, von großer Bedeutung ist, wurde zudem ein Vorgehensmodell entwickelt. Dieses ist nachfolgend beschrieben und soll die Verwendung der Modellierungstechnik unterstützen.

#### Vorgehensmodell

Das Vorgehensmodell gliedert den Entwicklungsprozess in unterschiedliche Phasen, siehe Abbildung 3.57. Zudem regelt es den Einsatz der Modellierungsmethoden. Im Projektvorhaben AutoVIBN wurde die frühe Entwicklungsphase, die zur Projektierung und Planung der Maschine dient, in eine Grob- und in eine Feinplanungsphase unterteilt. An diese schließt sich die eigentliche Konstruktion an, in welcher die disziplinspezifische Ausarbeitung erfolgt, was z. B. die CAD-Modelle beinhaltet.

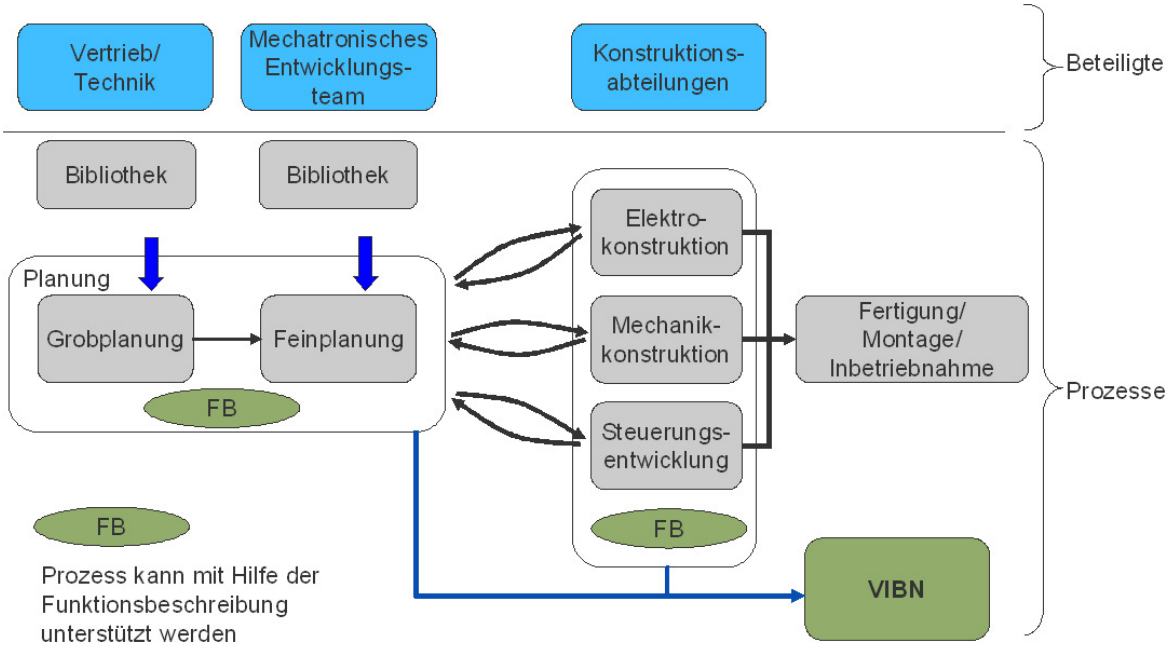



Abbildung 3.57: Vorgehen mit Hilfe der Funktionsbeschreibung

Die Grobplanungsphase dient zur Abstimmung mit dem Kunden. In Zusammenarbeit mit dem Vertrieb und Vertretern mit technischem Hintergrund des Herstellers können erste Mo-

delle erarbeitet werden, die sich zur firmenübergreifenden Kommunikation nutzen lassen. Hierbei gilt es, sowohl die Anforderungen des Kunden als auch die Gegebenheiten des Herstellers zu berücksichtigen. Dadurch lassen sich frühzeitig Missverständnisse vermeiden, wodurch zeitintensive Iterationsschleifen im Entwicklungsprozess vermieden werden können.

In der Feinplanungsphase werden die Ergebnisse der Grobplanung aufgegriffen und technische Lösungsstrukturen erarbeitet. Beteiligt ist idealer Weise ein mechatronisches Projektteam, das alle Disziplinen repräsentiert. Wie auch schon in der Feinplanungsphase kann hierdurch die Kommunikation verbessert werden. Eine interdisziplinäre Abstimmung unter Einbindung der gegebenen Randbindungen ist somit zu einem frühen Zeitpunkt realisierbar. Änderungen in den Teildisziplinen sollten am übergreifenden Funktionsmodell diskutiert werden, da sich hierdurch die Auswirkungen auf die anderen Bereiche simulieren und abschätzen lassen. Im Folgenden soll auf die einzelnen Phasen noch genauer eingegangen werden.

### Grobplanung

Da das Modell der Grobplanungsphase den Ausgangspunkt der Entwicklung einer Maschine oder Anlage darstellt, sollte dieses möglichst lösungsneutral erstellt werden. So ist es bspw. zu diesem Zeitpunkt noch nicht von Interesse, ob eine Sicherheitstüre von einem Pneumatikzylinder oder einem Elektromotor verfahren wird. Die Bewegung der Tür kann mit Hilfe eines hybriden Automaten abstrakt dargestellt werden. Neben der Definition der Endlagen ist dabei lediglich das Bewegungsverhalten durch die Differentialgleichungen auszudrücken.

Die Komponentenstruktur wird ebenso vereinfacht berücksichtigt, wie dies auch für die Geometrie der Maschine der Fall ist, siehe Abbildung 3.58. Bei der Modellierung stehen die Funktionen im Vordergrund und nicht die technische Lösung. Der Einfluss der Aktoren und der Sensoren wird stark auf die funktionalen Aspekte reduziert. Bereits in diesem Stadium ist eine Simulation möglich, um Eigenschaften analysieren zu können.

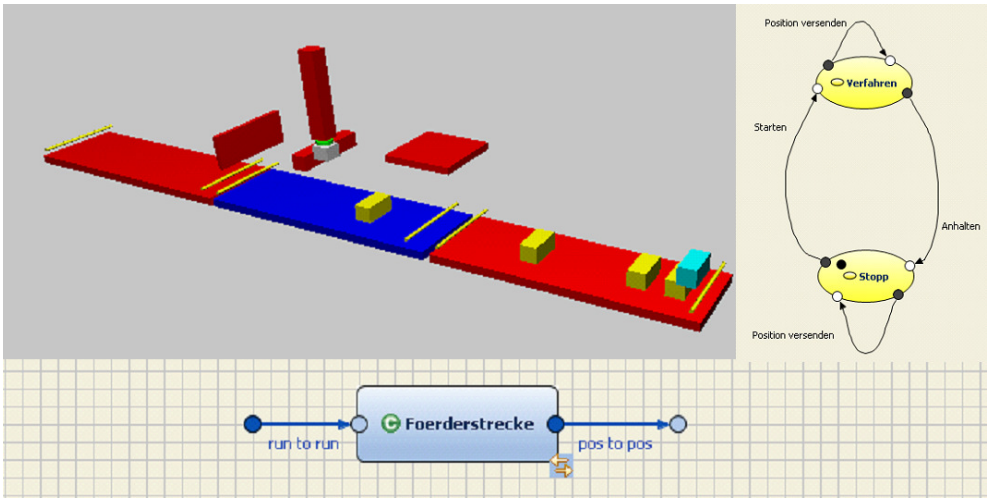



Abbildung 3.58: Modellierung in der Grobplanungsphase

### Feinplanung

Die Feinplanungsphase greift die Ergebnisse der Grobplanung auf. Gemäß deren Vorgaben ist das Funktionsmodell durch die Entwickler zu detaillieren. Hierbei ist die Komponentenstruktur zu erweitern und die wesentlichen technischen Elemente sind abzubilden. Desgleichen ist das Verhalten der Aktorik und Sensorik zu verfeinern und deren spezifische Charakteristika sind in die Modelle zu integrieren, siehe Abbildung 3.59. Im Rahmen der Feinplanungsphase wird zudem das Technische Ressourcenmodell angelegt, welches die Kopplung zur Steuerung ermöglicht.

Im Zuge der Feinplanungsphase ist ebenso die Geometrie der Anlage zu detaillieren. Dieser Schritt wird häufig auch in der industriellen Praxis durchgeführt, mit Hilfe grober CAD-Modelle werden erste Muster einer neuen Maschine erstellt. Im Rahmen des hier vorgeschlagenen Vorgehens kann dazu die Funktionsbeschreibung genutzt werden. Dies hat den Vorteil, dass dadurch schon in einer frühen Entwicklungsphase neben der Geometrie auch das Verhalten mit berücksichtigt werden kann.

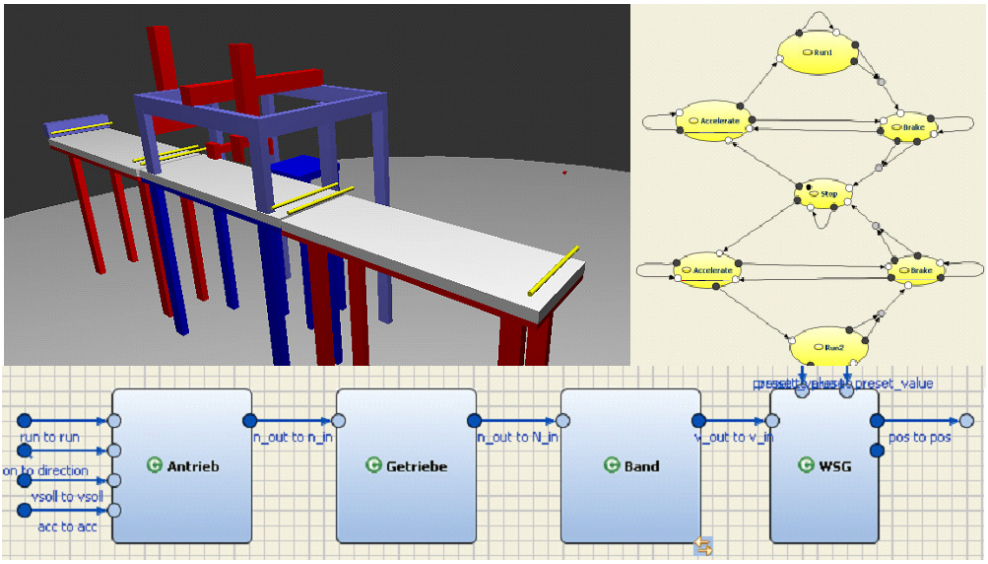



Abbildung 3.59: Modellierung in der Feinplanungsphase

## 3.8.2 Werkzeugunterstützung

Zur Unterstützung des entwickelten Modellierungsansatzes wurde ein Softwarewerkzeug entwickelt, mit welchem die Validierung der Methode durchgeführt werden konnte. Es handelt sich dabei jedoch um ein Funktionsmuster, welches in dieser Form nicht für den industriellen Einsatz geeignet ist. Eine Entwicklung zum Serienprodukt muss außerhalb des hier beschriebenen Vorhabens stattfinden. Im Folgenden wird auf ausgewählte Aspekte des Werkzeuges eingegangen.

### Modellierung von Funktionen

Die Modellierungstechnik an sich wurde bereits in AP2 näher erläutert, im hier vorliegenden Abschnitt soll deren Umsetzung in der Entwicklungsumgebung beschrieben werden. Zur Projektierung von Maschinen steht ein Editor zur Verfügung, der eine zum größten Teil grafische Modellierung erlaubt, siehe Abbildung 3.60. Alle Elemente der Modellierungstechnik, wie bspw. die Komponenten oder die hybriden Automaten, sind im Funktionsmuster hinterlegt. Für die Darstellung der Geometrie können in frühen Entwicklungsphasen Volumenprimitive verwendet werden. Auch ein Import von VRML-Dateien ist möglich.

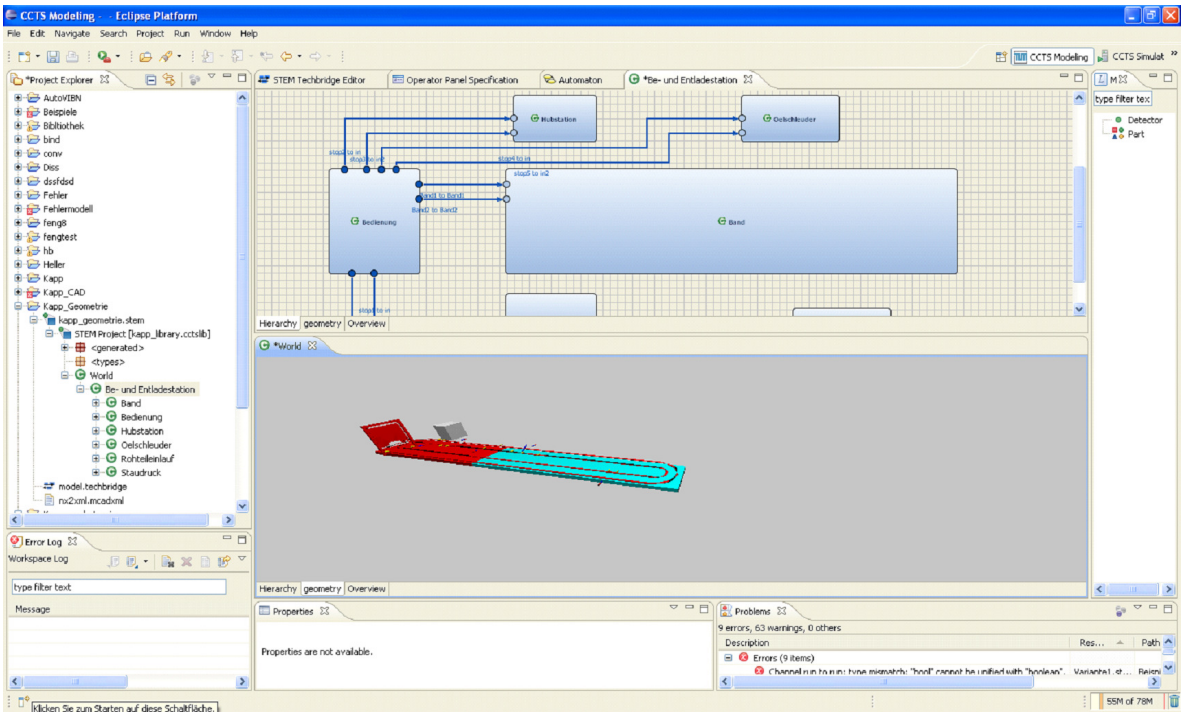



Abbildung 3.60: Funktionsmuster zur Unterstützung der Entwicklungstätigkeiten

### Simulation in frühen Entwicklungsphasen

Die projektierte Anlage kann direkt im Editor simuliert werden, siehe Abbildung 3.61. Voraussetzung hierfür ist, dass neben dem Maschinenverhalten auch die Steuerung berücksichtigt wurde. Es handelt sich dabei um eine ereignisbasierte Simulation, bei der Rückmeldungen, bspw. das Auslösen von Sensoren, Einfluss auf den Ablauf haben. Der Entwickler wird zudem durch vielfältige Darstellungsformen unterstützt. Neben der 3D-Visualisierung wären als Beispiel die Kontrollzustände der Automaten oder sog. Weg-Zeit-Diagramme zu nennen.

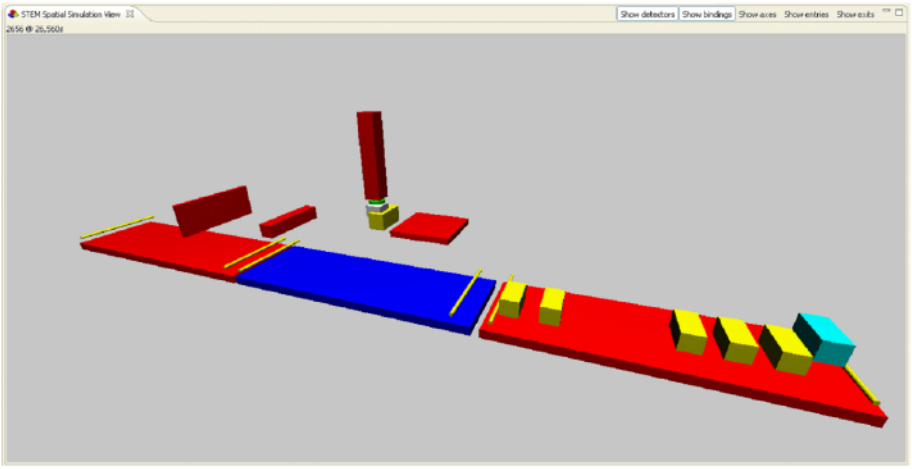



Abbildung 3.61: Simulation in der Funktionsbeschreibung

### Virtuelle Inbetriebnahme

Hinsichtlich der VIBN wird die für die Visualisierung und die Bedienung des Simulationsmodell ebenfalls auf das Funktionsmuster zurückgegriffen. Der Simulationskern wird in C++ ausgeleitet und verbindet sich mit der Entwicklungsumgebung über das Netzwerkprotokoll TCP/IP. Dieses Vorgehen hat den Vorteil, dass Simulation und Visualisierung auf getrennten Computern ablaufen können und sich somit die Zeitschrittweiten für die Berechnung reduzieren. In Abbildung 3.62 wird ein VIBN-Modell dargestellt, welches im entwickelten Softwarewerkzeug simuliert wurde.

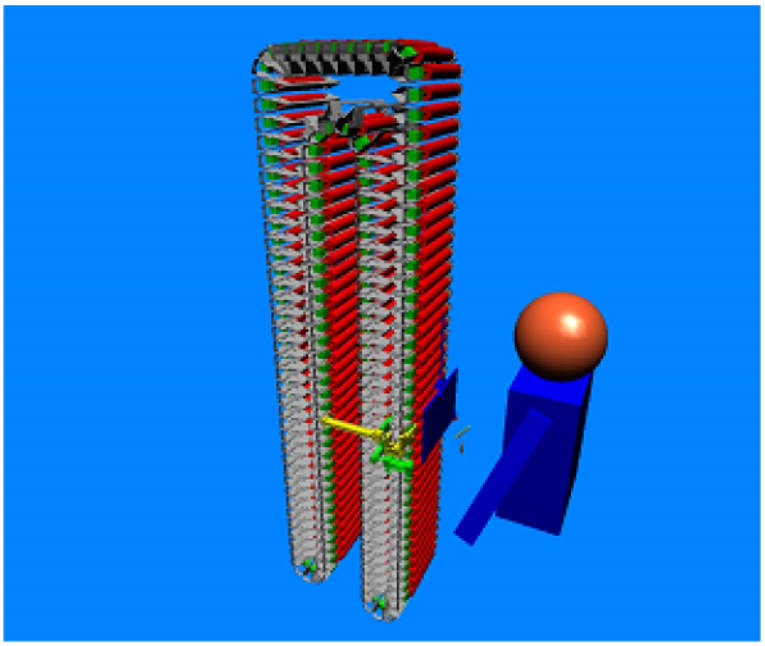


Abbildung 3.62: HIL-Modell in der Funktionsbeschreibung

### 3.8.3 Anwendungsbeispiele

Im Rahmen des Projektes wurden vier eigene sowie zwei industrielle Anwendungsbeispiele untersucht. Die ersten vier Szenarien waren unterschiedliche Förderbandsystem, die im Folgenden mit Variante 1 bis 4 bezeichnet werden. Die industriellen Anwendungsbeispiele wurden in Zusammenarbeit mit den Firmen Heller Machines und Kapp bearbeitet. Dabei handelte es sich im Falle der Firma Heller um das sog. hauptzeitparallele Rüsten eines Fräsbearbeitungszentrums. Die Firma Kapp stellte ein Be- und Entladesystem für eine Zahnradschleifmaschine als Versuchsobjekt zur Verfügung. Die Funktionen der Varianten 1 bis 4 und deren konstruktive Anforderungen wurden bereits in AP1 erläutert. Dabei wurden auch erste Konstruktionszeichnungen vorgestellt, die als Grundlage zur Abstimmung mit dem projektbegleitenden Ausschuss gedient haben. An dieser Stelle sollen nun die detaillierten und fertig konstruierten Versionen gezeigt werden.

#### Variante 1 und 2

Die Aufgabe der ersten beiden Varianten bestand darin, einen zufällig verteilten Materialflussstrom zu gruppieren und für gleiche Abstände zwischen dem Stückgut zu sorgen (Variante 1). Variante 2 wurde um einen Vereinzelungsprozess ergänzt. Hierfür wurden hintereinander geschaltete Förderbänder verwendet, die alle drehzahlgeregelt ausgeführt sind. Für die Modellierung wurden die Bänder als hierarchische Komponenten angelegt, die jeweils aus dem Band und zwei Lichtschranken bestehen. Abbildung 3.63 zeigt die zugehörigen Modelle.

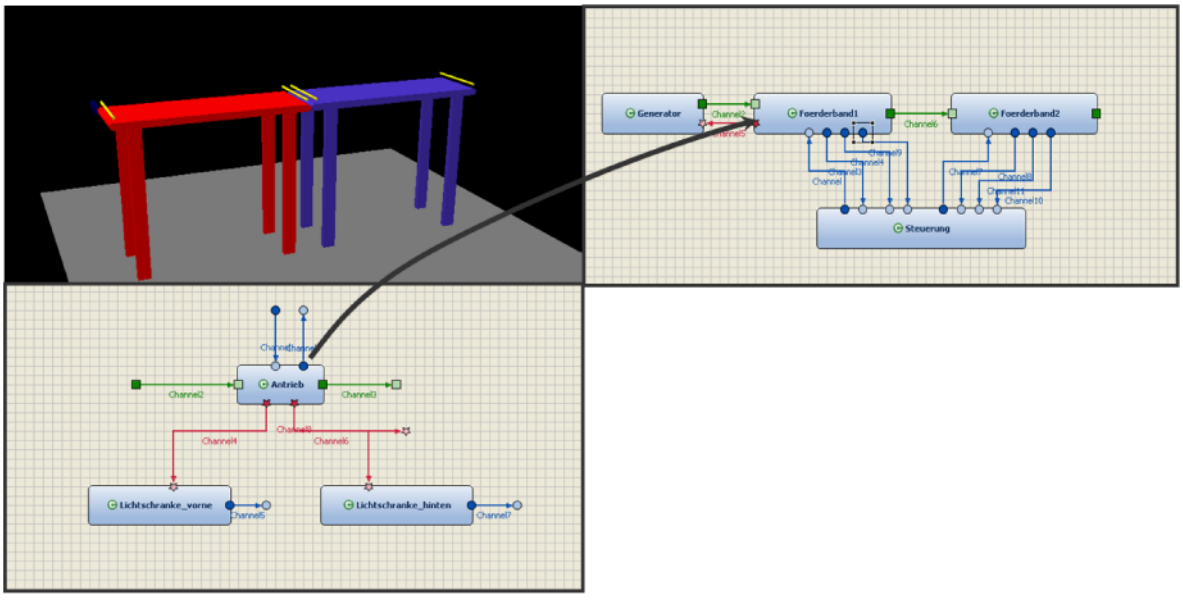



Abbildung 3.63: Modellaufbau von Variante 1

Nachdem in der Funktionsbeschreibung ein grobes 3D-Modell angelegt wurde, erfolgte die Übernahme von Details aus der mechanischen Konstruktion bzw. dem CAD-Modell. Das

detaillierte MCAD-Modell ist in Abbildung 3.64 zu sehen.

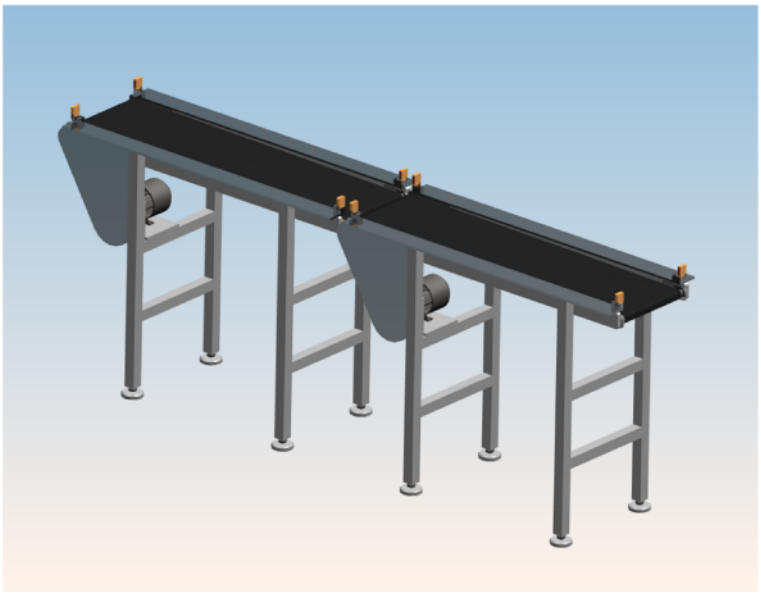


Abbildung 3.64: MCAD-Modell der Variante 1

Die Variante 2 unterscheidet sich von Variante 1 dadurch, dass für den Vereinzelungsprozess ein drittes Förderband in der Mitte eingefügt wurde. Dieses trennt die Materialflussobjekte voneinander und stoppt anschließend, um das Stückgut vom Band greifen zu können. Diese Funktion ist vor allem für die Varianten 3 und 4 entscheidend. Abbildung 3.65 zeigt den Entwurf, der mit dem projektbegleitenden Ausschuss erarbeitet wurde.

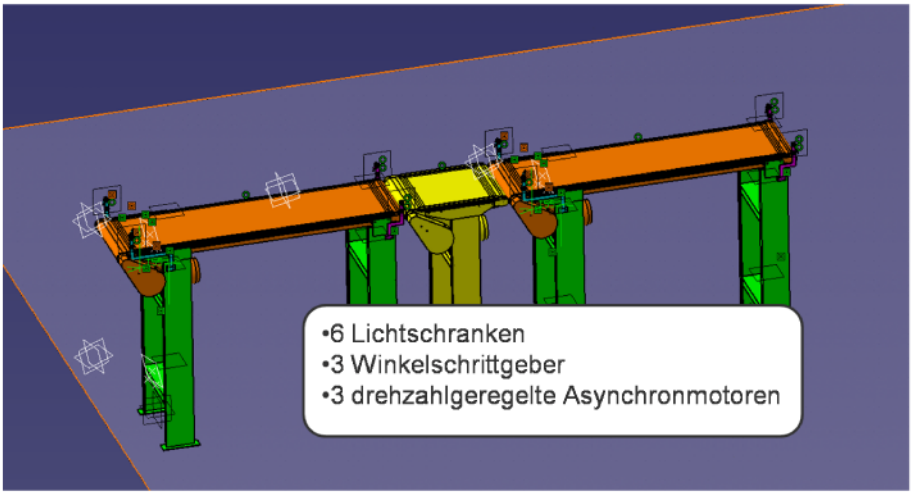



Abbildung 3.65: Prinzipbild der Variante 2

Auch bei Variante 2 wurden anschließend Daten aus dem CAD ins Modell übernommen. 3.66 zeigt das detaillierte MCAD-Modell.

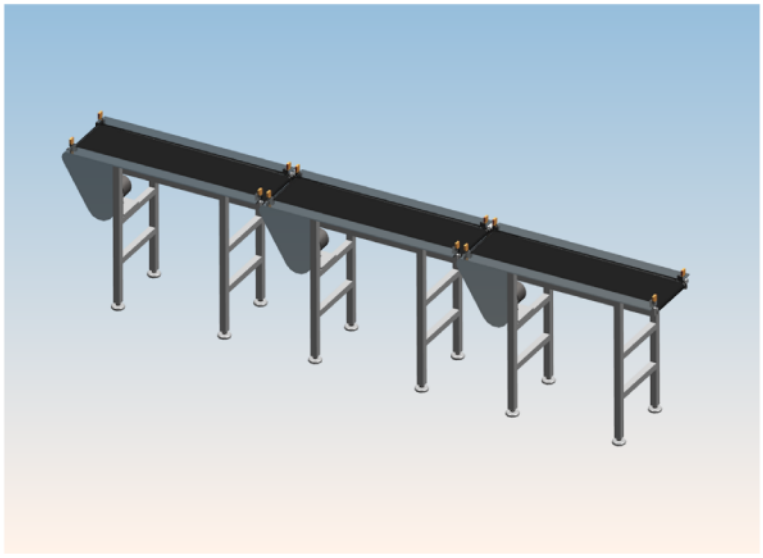


Abbildung 3.66: MCAD-Modell der Variante 2

Um die Datenkonsistenz nicht nur zur mechanischen Konstruktion, sondern auch zur elektrischen Konstruktion zu waren, wurde ein Abgleich der Komponenten mit dem ECAD durchgeführt. Mit Hilfe einer Programmierschnittstelle wurden die Daten aus dem ECAD-Modell in eine XML-Datei überführt. Diese kann dann anschließend in der Funktionsbeschreibung eingelesen und mit dem bestehenden Modell verglichen werden. Abbildung 3.67 zeigt einen Auszug aus dem erstellten Schaltplan für Variante 1 und 2.

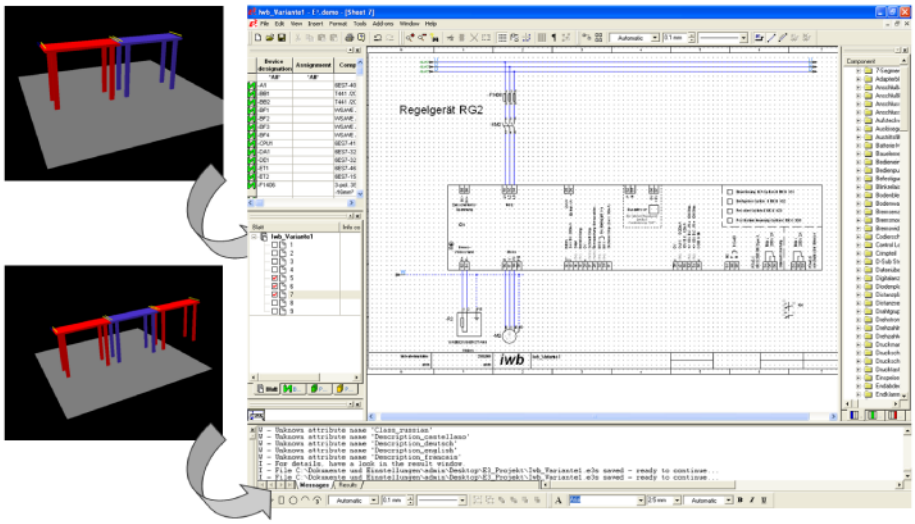



Abbildung 3.67: Variante 1 und 2 ECAD-Import

Die Ergebnisse aus Variante 1 und 2 können mit dem Aufbau zweier materialflussintensiver Modelle und dem zugehörigen Simulationslauf umschrieben werden. Mit Hilfe dieser Erkenntnisse wurden anschließend die Varianten 3 und 4 aufgebaut, die über eine zusätzliche Achskinematik verfügen.

### Variante 3 und 4

Diese beiden Varianten bauen auf den vorangegangenen Modellen auf. Die Variante 3 unterscheidet sich dabei von Variante 4 lediglich in der Ausbautiefe. Daher wird im Folgenden nur Variante 4 beschrieben. Die Funktion dieser Variante ist, Objekte eines Materialflussstroms zu vereinzeln, mit Hilfe einer Dreiachskinematik samt Drehgreifer vom Band zu nehmen, zu drehen und auf einem SPC-Arbeitsplatz abzustellen. Dort soll dieses Objekt gemessen und der Anlage wieder zugeführt werden. Im Anschluss an die Portalstation wird das Bauteil dann entweder im Materialfluss belassen oder mit Hilfe eines Abschiebers quer vom Band geschoben. Der Eingriff kann somit entweder vom Bediener bestimmt oder automatisch bei jedem beliebigen Teil vorgenommen werden. Abbildung 3.68 stellt diese Funktionen dar.

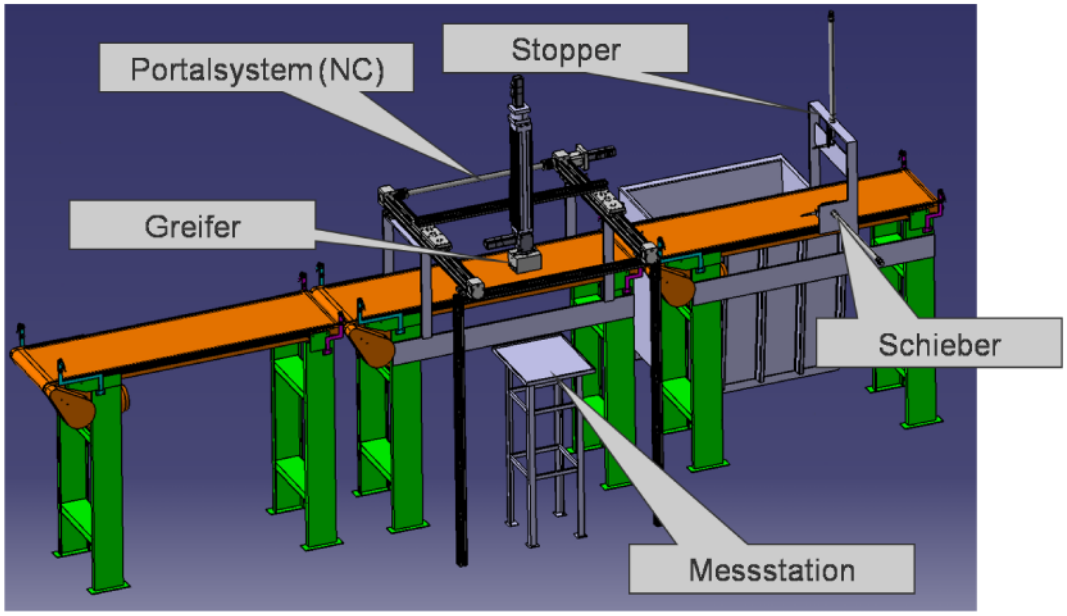



Abbildung 3.68: Prinzipbild der Variante 4

Die Modellerstellung erfolgte im Editor für die Funktionsbeschreibung. Die besonderen Herausforderungen der vierten Variante waren die Abbildung des Greifers sowie der Drehbewegung der Achsen. Weiterhin wurde ein HMI-Modell eingeführt, das es ermöglicht, Bedienbilder anzulegen und damit das Modell zu steuern. Dies war nötig, damit die Achsen manuell im Einrichtbetrieb verfahren werden konnten. Abbildung 3.69 zeigt das detaillierte MCAD-Modell.

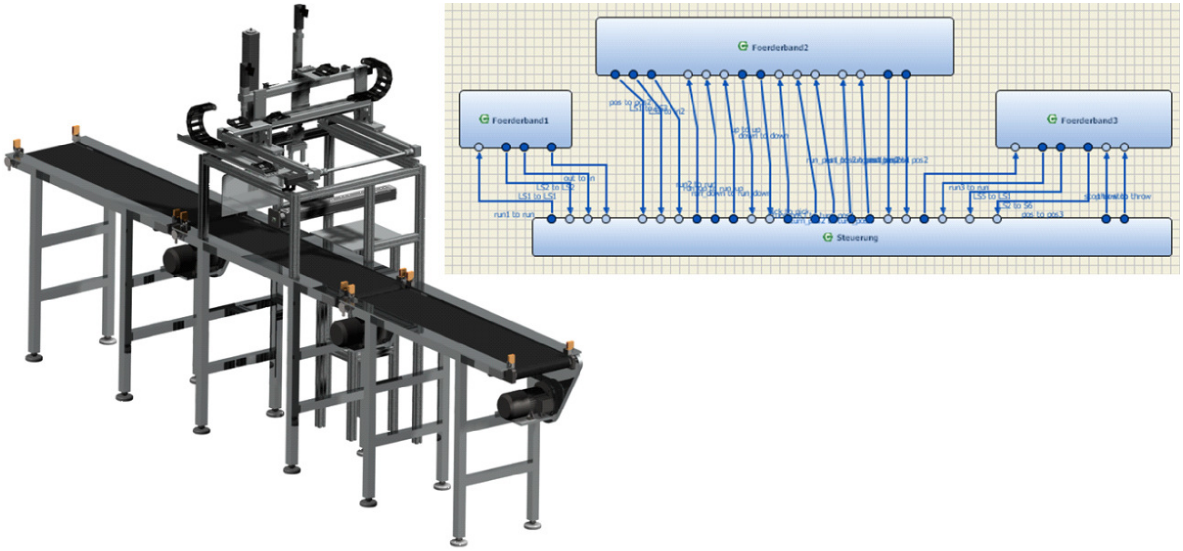



Abbildung 3.69: Modellaufbau Variante 4

Die Ergebnisse aus den vier Varianten können folgendermaßen zusammengefasst werden:

- Entwicklung, Konstruktion und Simulation verschiedener Förderbandsysteme
- Test der Modellierungstechnik und des Funktionsmusters
- Validierung der Vorgehensweise
- Erstellung und Test der Technischen Ressourcenmodelle
- Kopplung an die reale Steuerung
- Test in Hardware-in-the-Loop-Anordnung

Die einfachen Fallbeispiele dienten vornehmlich zur Prüfung und zur Validierung der hier vorgestellten Methode. Auf dieser Basis konnten dann die komplexeren Szenarien der von den Herstellern zur Verfügung gestellten Beispiele aufgebaut werden.

### Hauptzeitparalleles Rüsten

Als reales Anwendungsbeispiel wurde zunächst ein Ausschnitt einer Werkzeugmaschine, das sog. hauptzeitparallele Rüsten, angegangen. Hierzu stellte die Firma Heller die nötigen Daten zur Verfügung und unterstützte zudem bei der Umsetzung der HIL-Simulation. Die Wahl fiel auf einen Maschinenabschnitt, für welchen bereits ein Simulationsmodell seitens Heller besteht. Somit konnte der hier entwickelte Ansatz direkt mit herkömmlichen Methoden verglichen werden. Abbildung 3.70 zeigt die Gesamtmaschine.


Abbildung 3.70: Hauptzeitparalleles Rüsten

Die Funktion des Rüstens der Werkzeuge kann parallel zum Bearbeitungsprozess vollzogen werden. Dazu verfügt die Maschine über eine Werkzeugkette mit einer festen Zahl (maschinenabhängig) von Werkzeugaufnahmen. Um einen neuen Auftrag werkzeugseitig zu rüsten, geht der Bediener folgendermaßen vor. Er wählt zuerst ein Werkzeug mit einer Platznummer an und lässt es zum Ausgabeschacht befördern. Danach setzt er das neue Werkzeug ein und spezifiziert über ein HMI, um welches Teil es sich handelt. Mit der Quittierung wird das Werkzeug über eine angetriebene Kette zu dem vorgesehenen Platz befördert. Die wesentlichen Elemente zum Werkzeugwechsel sind dabei die Kette selbst, eine Indexiereinrichtung, ein Ausschieber und eine automatische Schutztüre, die auch manuell bedient werden kann. Endlagensensoren und Absolutwertgeber an der Kette sorgen zu jeder Zeit für die Erfassung der Istzustände der Arbeitszylinder und der Kettenposition. Der Modellaufbau wurde basierend auf den Erkenntnissen aus den Varianten 1 bis 4 hierarchisch angelegt. Es soll an dieser Stelle auf ein Detailproblem eingegangen werden, dass sich sowohl beim Aufbau des Modells der Firma Heller als auch bei dem der Firma Kapp als Herausforderung darstellte. Bisher waren bewegte Materialobjekte lediglich Komponenten, die sich auf einem Band bewegten oder in einem Greifer über mehrere Achsen im Raum platziert werden konnten. Im vorliegenden Beispiel wurden jedoch 160 Werkzeughalter über eine komplexe Kettenbewegung verfahren. Dies erforderte die Erweiterung des Mechanismus zur Achsdefinition, wie er bereits in AP2 vorgestellt wurde, siehe auch Abbildung 3.71. Zudem war der Materialfluss des Anwendungsbeispiels dreistufig (Magazinplatz, Werkzeughalter, Werkzeug), was ebenfalls zu Verbesserungen am Meta-Modell genutzt wurde.

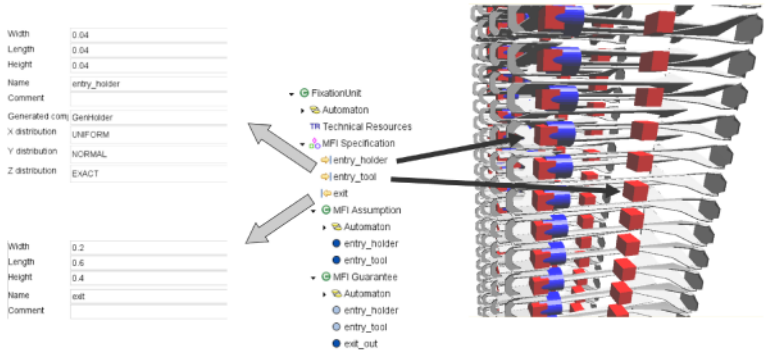



Abbildung 3.71: Materialmodellierung Heller

Die Steuerung der Maschine basiert auf einer Siemens Sinumerik 840D. In Abbildung 3.72 lässt sich das Bedienfeld und die Steuerung erkennen. Diese wurde über den Profibus mit einer SIMBProPCI-Simulationskarte der Firma Siemens gekoppelt, auf die das Abbild der Profibussstruktur aufgespielt wurde. Die Simulation der Antriebe, der Materials und der Sensoren läuft als eigenständige Anwendung im C++-Simulationskern, der auf dem Rechner in Abbildung 3.72 arbeitet.

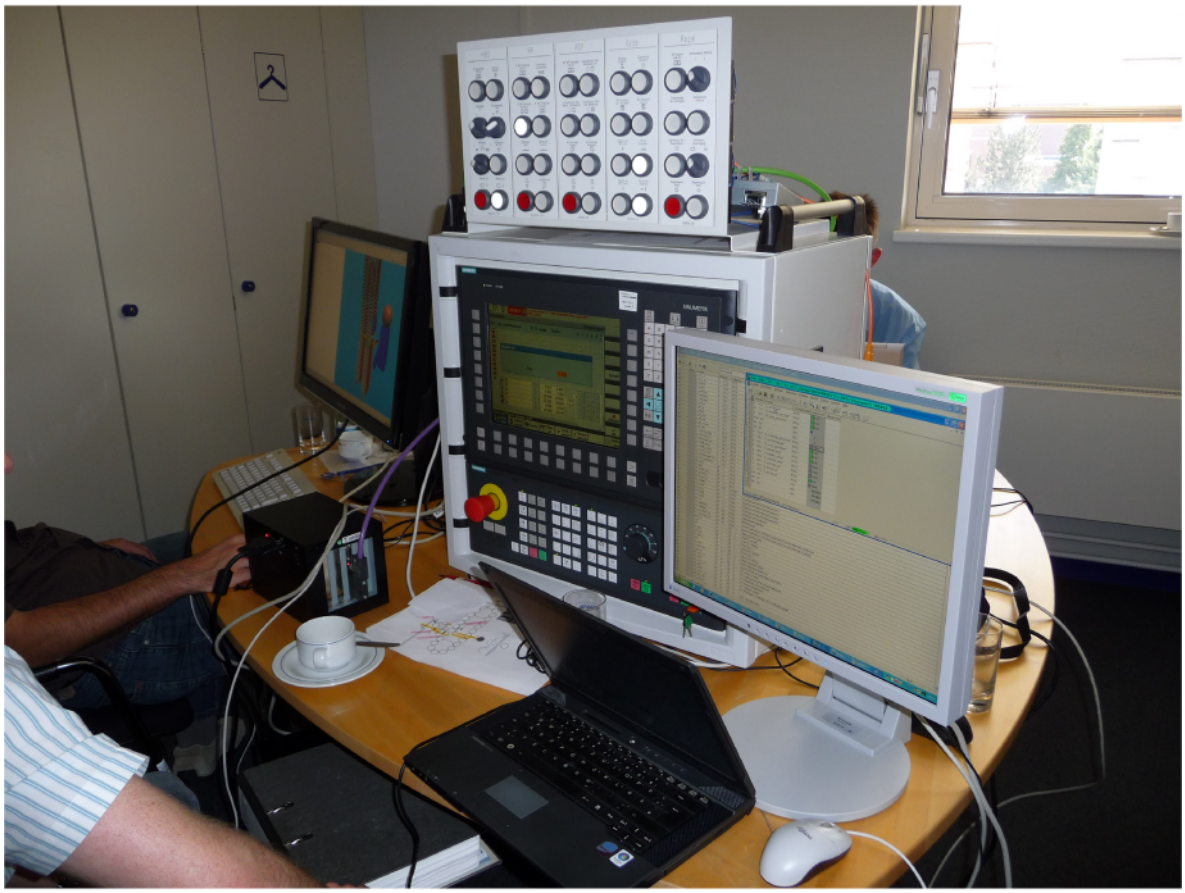


Abbildung 3.72: Aufbau des Versuchsstandes bei der Firma Heller

Im Vergleich mit dem bereits bei der Firma Heller existierenden Simulationsmodell zeigt der hier beschriebene Ansatz eine wesentlich größere Detailtreue. Dies manifestiert sich vor allem daran, dass alle Kettenelemente (160) berücksichtigt wurden. Zudem verfahren die Objekte im Raum, sodass hinsichtlich der Simulation die exakte Position Einfluss auf das Geschehen hat. Durch die integrierte Kollisionserkennung konnten bspw. Näherungssensoren sehr einfach und schnell modelliert werden. Insgesamt war der Modellierungsaufwand deutlich geringer, da die Funktionsbeschreibung auf einem abstrakteren Informationsniveau erstellt wird. Die Ergebnisse werden grafisch in Abbildung 3.73 zusammengefasst.

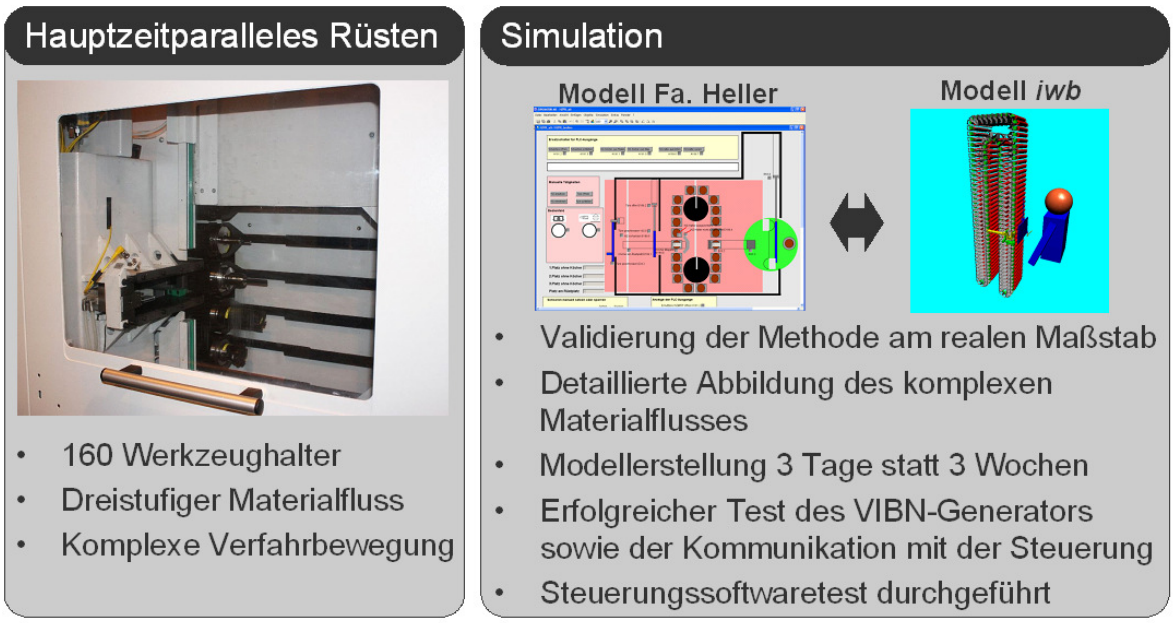



Abbildung 3.73: Vergleich des Simulationsansatzes der Firma Heller mit dem des Projektes AutoVIBN

**Be- und Entladessystem**

Das Be- und Entladesystem der Firma Kapp besteht aus einem Rundlauf mit 30 Paletten und hat die Aufgabe einer Schleifmaschine Teile zuzuführen und diese nach der Bearbeitung auszuschleusen. Das Palettentransportband besteht aus einem geschütztem und einem offenen Bereich. Abgeschirmt ist der Einlauf der Rohteile bis zum Ausgang der Ölschleuder. Die nicht bearbeiteten Teile werden vom Bediener im offenen Bereich aufgelegt, im Schutzbereich bearbeitet und von Öl befreit. Danach werden die Fertigteile wieder zum Bediener verfahren. Das Band dreht sich im Uhrzeigersinn. Die Werkstücke werden mittels einer mechanischen Konstruktion am Werkstückträger (Palette) auf Roh- oder Fertigteil detektiert. Eine weitere mechanische Abfrage erfolgt durch eine Konturmaske (Kulisse) am Einlauf. Diese lenkt aus und stoppt somit das Band, falls ein Fertigteil oder ein falsches Teil erneut in den Kreislauf einfahren würde. Bei der Übergabe an die Schleifmaschine sorgt eine Hubstation und ein Ringlader mit zwei Aufnahmen für die Übergabe der Werkstücke. Die Fertigteile wer-

den anschließend vom Öl gesäubert und in den offenen Bereich zurückgefahren. Abbildung 3.74 stellt die Anlage als Layout und die wesentlichen Stationen dar.

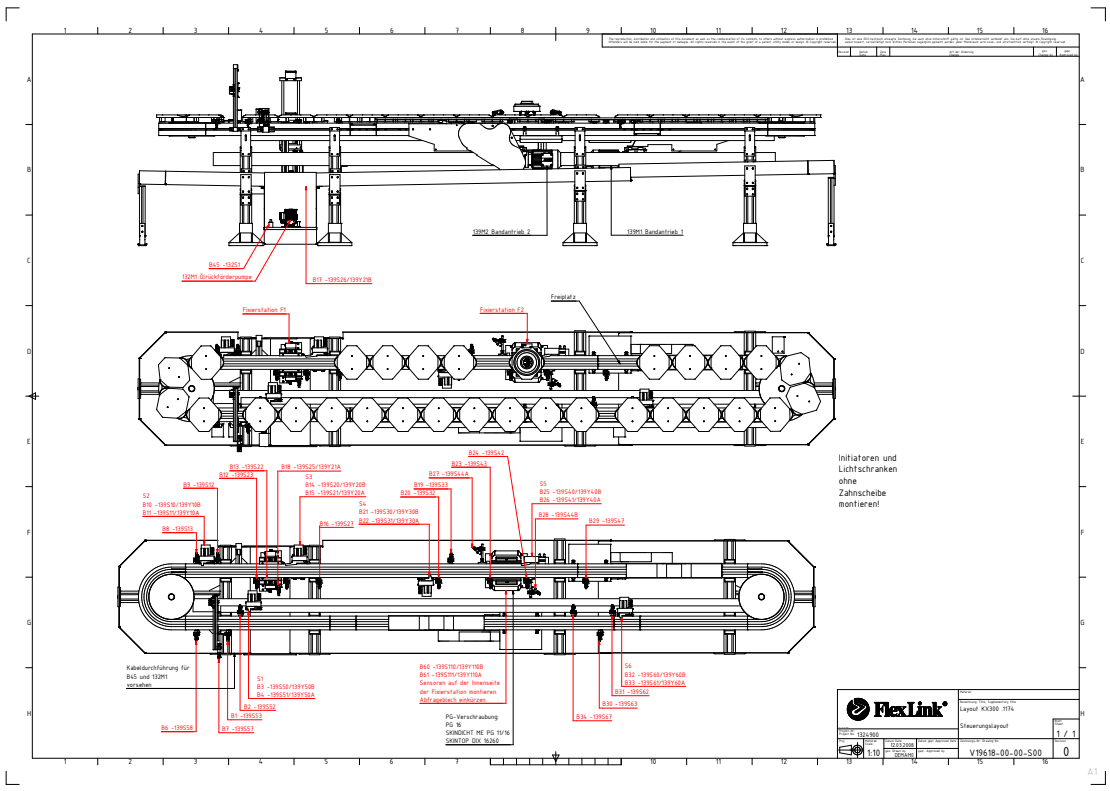



Abbildung 3.74: Layout des Be- und Entladesystems

Wie auch schon im vorangegangenem Anwendungsbeispiel musste ein komplexer und mehrstufiger Materialfluss abgebildet werden. Besonders die Rohteilerkennung musste im Zuge der Simulationstätigkeiten modifiziert und verbessert werden. Hinsichtlich der Strukturierung des Simulationsmodells wurde ein mechatronischer Ansatz verfolgt. Es wurden Einheiten mit Maschinenverhalten und Steuerungsintelligenz gebildet, die eine Wiederverwendung erleichtern sollen. Zudem kann dann die Kopfsteuerung im Modell wesentlich kompakter ausfallen. Als wesentliche Elemente sind im Modell folgende Bestandteile enthalten:

- Band
- Bedienung
- Hubstation
- Ölschleuder
- Rohteileinlauf
- Staudruck

In Abbildung 3.75 und Abbildung 3.76 ist die Anlage dargestellt. Mit Hilfe des Kollisionsmodells konnte somit auf eine einfache Art und Weise die Geometrieprüfung am Rohteileinlauf

umgesetzt werden.

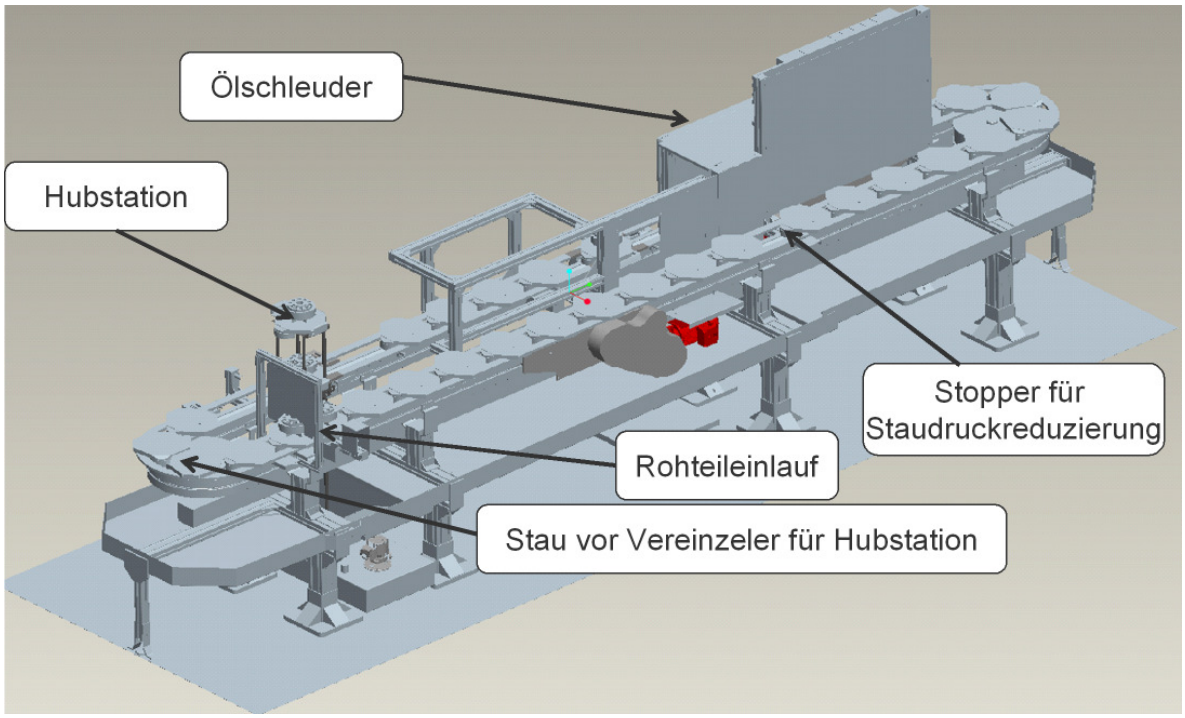



Abbildung 3.75: MCAD-Modell der Anlage

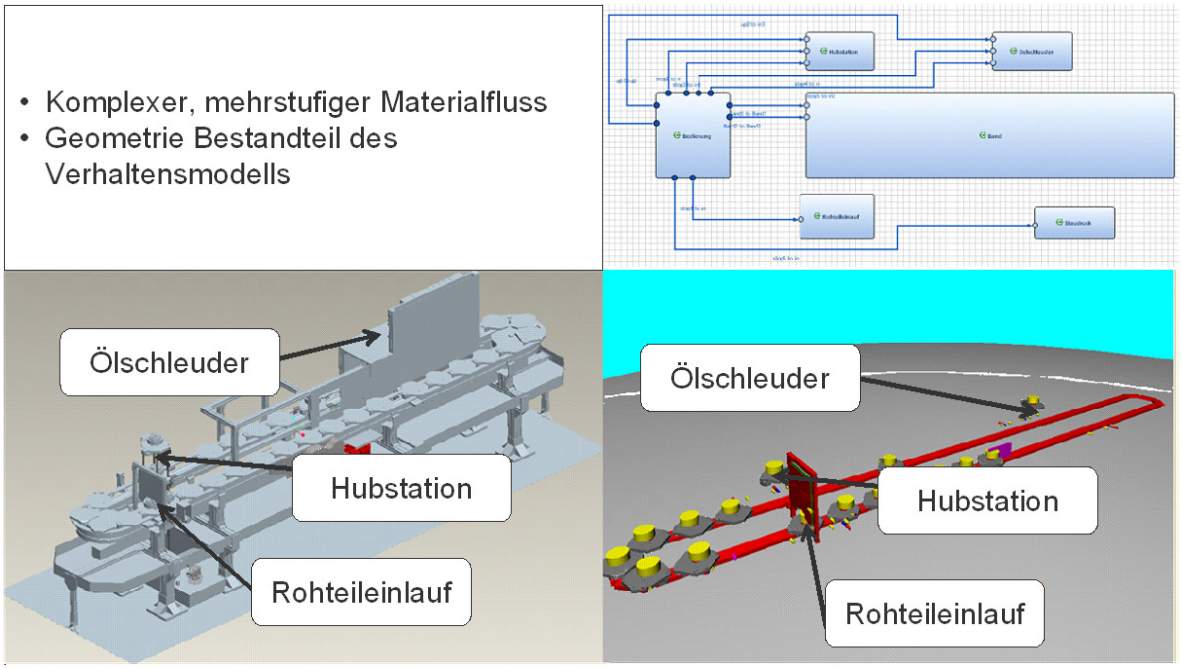



Abbildung 3.76: Simulationsmodell der Anlage

Wie bereits erwähnt verfügt die Maschine über einen mechanischen Ansatz zur Erkennung von Rohteilen. Dieser ist mit Hilfe eines Kipphebels realisiert, der über einen Näherungssensor am Rohteileinlauf abgefragt wird. Vom Bediener ist dieser Hebel beim Auflegen von Teilen zu betätigen. Dies hat zu Modifikationen am Meta-Modell der Funktionsbeschreibung geführt, wodurch auch dieser Sachverhalt abgebildet werden konnte siehe Abbildung 3.77.

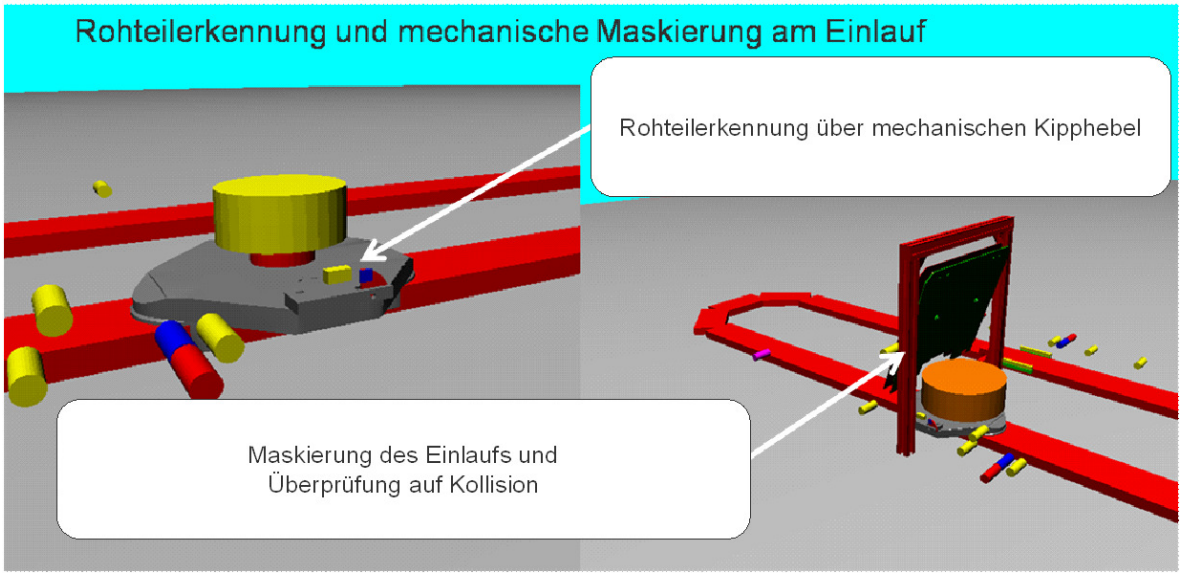



Abbildung 3.77: Rohteilerkennung

Nach dem Aufbau der Funktionsbeschreibung und der Einbindung der CAD-Daten konnte das VIBN-Modell abgeleitet werden. Dieses wurde dann über eine HIL-Kopplung mit der realen Steuerung verbunden. Anschließend konnten Testläufe im Automatikbetrieb durchgeführt werden. Der Aufwand, um eine bestehende Steuerung (in diesem Fall eine Sinumerik 840D samt zugehöriger SPS) mit einem Modell zu verbinden, soll an dieser Stelle kurz beschrieben werden. Zunächst muss darauf hingewiesen werden, dass im Simulationsfall selten eine komplette Anlage, sondern meist ein Ausschnitt davon modelliert wird. Das Steuerungssystem lässt sich in der Regel aber nicht auf eine triviale Art und Weise isolieren, sodass nur der Teil aktiv ist, der simuliert werden soll. Demzufolge muss hier zusätzlicher Aufwand betrieben werden, um alle anderen Stationen einer Anlage steuerungsseitig passiv zu schalten. Dies ist vor allem für den Automatikbetrieb nötig, da hier bestimmte Signale zwischen den Stationen ausgetauscht werden. Hierzu muss eine Reihe von Bedingungen aller Stationen erfüllt sein. Weiterhin sind alle Achsen, die nicht im Simulationsmodell abgebildet werden, aus der Hardwarekonfiguration zu löschen. Dieses reduzierte Steuerungsprojekt muss mit der Struktur auf der Hardwaresimulationskarte (SIMBAProPCI) übereinstimmen, da sonst die Steuerung nicht in den RUN-Modus übergeht und Busfehler meldet. Zudem müssen in der Schrittkettenabarbeitung der Programme diejenigen Schritte ausgelassen werden, die nicht durch das Modell erfasst werden. Abschließend soll noch ein Auszug aus der Simulation gezeigt werden, der in Abbildung 3.78 dargestellt ist.

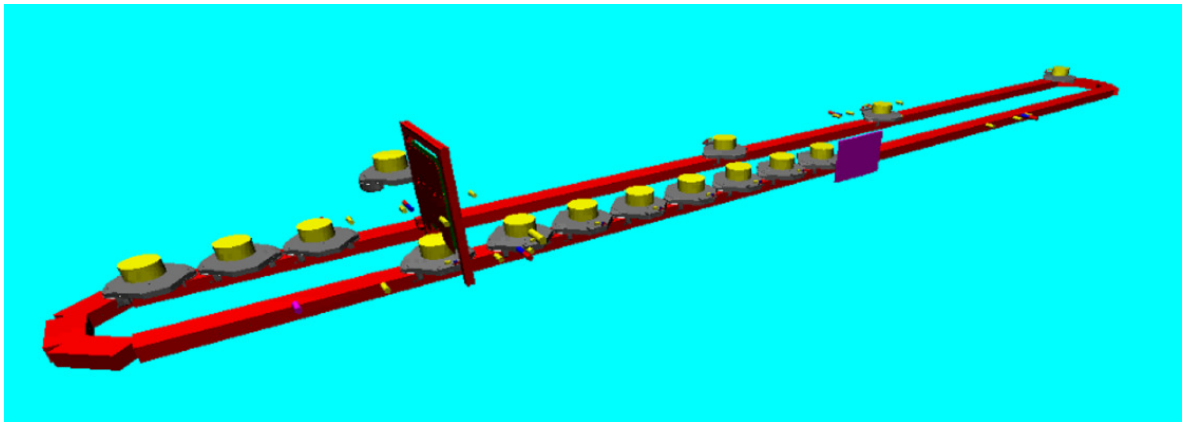


Abbildung 3.78: Auszug aus der Simulation des Be- und Entladesystems der Firma Kapp

## 3.9 AP10: Kritische Bewertung der erzielten Forschungsergebnisse

Die im Forschungsvorhaben AutoVIBN erzielten Ergebnisse wurden fortlaufend durch den projektbegleitenden Ausschuss bewertet. Dadurch sollte eine praxisnahe Gestaltung des Ansatzes gewährleistet werden. Zudem sollte auf diese Art und Weise der Transfer in die Wirtschaft erleichtert werden.

Anbei sind die Bewertungsergebnisse des Ausschusses dargestellt. Als Grundlage dienen die Schulnoten, womit sich folgende Einteilung ergibt:

- 1 = sehr gut
- 2 = gut
- 3 = befriedigend
- 4 = ausreichend
- 5 = mangelhaft
- 6 = ungenügend

Die Noten werden für die Projektergebnisse, die Projektbearbeitung, die Ergebnispräsentation und die Eigeninitiative vergeben. Aus der ermittelten Gesamtnote leitet die Forschungsvereinigung Werkzeugmaschinen folgende Maßnahmen ab:

- 1,0 bis 2,7: Projektverlauf i.O.
- 2,7 bis 3,3: Betreuung intensivieren
- 3,3 bis 4,0: Intervention des Projektpaten
- ab 4,0: Gespräch mit der Institutsleitung

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum: | 27.02.2008 | | | | |
| | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| 1 | 3 | 3 | 3 | 2 | 2,75 |
| 1 | | 1 | 1 | 2 | 1,33 |
| 1 | 2 | 2 | 2,5 | 2,5 | 2,25 |
| 1 | | | 2 | 1,5 | 1,75 |
| 1 | 3 | 2 | 2 | 3 | 2,50 |
| 1 | 3 | 2 | 3 | 2 | 2,50 |
| 1 | 1 | 1 | 1 | 1 | 1,00 |
| 1 | | 2 | 1,5 | 1 | 1,50 |
| 8 | 2,40 | 1,86 | 2,00 | 1,88 | |
| | | | | | 1,95 |
| | Projektverlauf i.O. | | | | |

Abbildung 3.79: Bewertung des Kick-Off-Treffens am 27. Februar 2008

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum: | 11.06.2008 | | | | |
| | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| 1 | 2 | 2 | 1 | 1 | 1,50 |
| 1 | 1 | 1,5 | 1 | 1 | 1,13 |
| 1 | 1,5 | 1,5 | 1,5 | 1,5 | 1,50 |
| 1 | 2 | 1 | 1 | 1 | 1,25 |
| 1 | 1,5 | 1,5 | 1,7 | 1,6 | 1,58 |
| 1 | 2 | 2 | 1 | 1 | 1,50 |
| 1 | 1 | 1,5 | 1 | 1 | 1,13 |
| 1 | 2 | 2 | 2 | 1,5 | 1,88 |
| 1 | 1,3 | 1,2 | 1,2 | 1,1 | 1,20 |
| 9 | 1,59 | 1,58 | 1,27 | 1,19 | |
| | | | | | 1,41 |
| | Projektverlauf i.O. | | | | |

Abbildung 3.80: Bewertung des Statustreffens am 11. Juni 2008

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum: | 02.10.2008 | | | | |
| | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| 1 | 2,0 | 2,0 | 1,0 | 1,0 | 1,50 |
| 1 | 2,0 | 2,0 | 2,0 | 1,0 | 1,75 |
| 1 | 2,0 | 2,0 | 1,0 | 1,0 | 1,50 |
| 1 | 1,0 | 1,0 | 1,0 | 1,0 | 1,00 |
| 1 | 1,0 | 2,0 | 2,0 | 1,0 | 1,50 |
| 1 | 1,5 | 1,5 | 1,5 | 1,0 | 1,38 |
| 1 | 2,0 | 1,0 | 1,0 | 1,5 | 1,38 |
| 0 | 2,0 | 2,0 | 1,0 | 2,0 | 1,75 |
| 1 | 2,0 | 1,0 | 2,0 | 1,0 | 1,50 |
| 1 | 2,0 | 1,5 | 1,5 | 1,0 | 1,50 |
| 9 | 1,94 | 1,78 | 1,56 | 1,28 | 1,64 |
| | Projektverlauf i.O. | | | | |

Abbildung 3.81: Bewertung des Statustreffens am 2. Oktober 2008

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum: | 27.01.2009 | | | | |
| | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| 1 | 2,0 | 1,5 | 2,0 | 1,5 | 1,75 |
| 1 | 2,5 | 1,5 | 3 | 1 | 2,00 |
| 1 | 2,0 | 1,0 | 2,0 | 1,0 | 1,50 |
| 1 | 2,0 | 1,9 | 2,1 | 1,8 | 1,95 |
| 1 | 2,0 | 2,0 | 2,0 | 1,0 | 1,75 |
| 1 | 2,0 | 1,5 | 2,3 | 1,5 | 1,83 |
| 1 | 1,8 | 1,3 | 2,4 | 1,3 | 1,70 |
| 7 | 2,04 | 1,53 | 2,26 | 1,30 | 1,78 |
| | Projektverlauf i.O. | | | | |

Abbildung 3.82: Bewertung des Statustreffens am 27. Januar 2009

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum:<br>Name: | 11.06.2008 | | | | |
| | | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| | 1 | 1,5 | 1,5 | 1,5 | 1,5 | 1,50 |
| | 1 | 1,5 | 1,5 | 1,6 | 1 | 1,40 |
| | 1 | 1,2 | 1 | 1,8 | 1 | 1,25 |
| | 1 | 2 | 1 | 2 | 1 | 1,50 |
| | 1 | 1 | 2 | 2 | 1 | 1,50 |
| | 1 | 1 | 1 | 2 | 1 | 1,25 |
| | 1 | 1,5 | 1,7 | 1,7 | 1,5 | 1,60 |
| | 1 | 2 | 2 | 2 | 1 | 1,75 |
| | 1 | 2 | 1,5 | 2 | 1,5 | 1,75 |
| | 1 | 2 | 1,5 | 1,5 | 2 | 1,75 |
| | 1 | 2 | 2 | 1 | 1 | 1,50 |
| | 1 | 1,2 | 1,4 | 1,8 | 1,3 | 1,43 |
| | 12 | 1,58 | 1,51 | 1,74 | 1,23 | |
| | | | | | | 1,51 |
| | | Projektverlauf i.O. | | | | |

Abbildung 3.83: Bewertung des Statustreffens am 5. Mai 2009

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum:<br>Name: | 17.09.2009 | | | | |
| | | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| | 1 | 2 | 1,5 | 1,5 | 2 | 1,75 |
| | 1 | 2 | 2 | 2 | 2 | 2,00 |
| | 1 | 1,5 | 1,5 | 2 | 1,5 | 1,63 |
| | 1 | 2 | 2 | 2 | 1 | 1,75 |
| | 1 | 2 | 2 | 2 | 2 | 2,00 |
| | 1 | 1,5 | 1,5 | 2 | 2 | 1,75 |
| | 1 | 2 | 2 | 1,5 | 1 | 1,63 |
| | 1 | 2 | 2 | 2 | 1 | 1,75 |
| | 1 | 1,7 | 1,5 | 1,6 | 1 | 1,45 |
| | 9 | 1,86 | 1,78 | 1,84 | 1,50 | |
| | | | | | | 1,74 |
| | | Projektverlauf i.O. | | | | |

Abbildung 3.84: Bewertung des Statustreffens am 17. September 2009

| Projekt: | AiF 274 ZN "Automatische Generierung von Verhaltensmodellen aus CAD_Daten für die qualitätsorientierte virtuelle Inbetriebnahme" | | | | |
|---|---|---|---|---|---|
| Datum:<br>Name: | 03.12.2009 | | | | |
| | | Projektergebnisse | Projektbearbeitung | Ergebnispräsentation | Eigeninitiative | |
| | 1 | 1,5 | 1,5 | 1 | 1,5 | 1,38 |
| | 1 | 1 | 2 | 2 | 1 | 1,50 |
| | 1 | 1 | 1 | 1 | 1 | 1,00 |
| | 1 | 1 | 1 | 1 | 1 | 1,00 |
| | 1 | 1,5 | 1,5 | 1,5 | 1,5 | 1,50 |
| | 1 | 2 | 1 | 2 | 1 | 1,50 |
| | 1 | 1 | 1 | 1 | 1 | 1,00 |
| | 7 | 1,29 | 1,29 | 1,36 | 1,14 | |
| | | | | | | 1,27 |
| | | Projektverlauf i.O. | | | | |

Abbildung 3.85: Bewertung des Statustreffens am 3. Dezember 2009

# 4 Zusammenfassung & Ausblick

## 4.1 Zusammenfassung

Das Ergebnis des Forschungsvorhabens AutoVIBN ist eine Methode, die die frühzeitige Einbindung der VIBN in den Entwicklungsprozess und die automatisierte Ableitung der dafür notwendigen Modelle erlaubt. Die Grundlage hierfür ist die sog. Funktionsbeschreibung, die eine abstrakte Modellierung des Steuerungs- und des Maschinenverhaltens ermöglicht. Diese Funktionsbeschreibung wurde als gemeinsames, zentrales Artefakt für den interdisziplinären Entwurf von mechatronischen Systemen konzipiert. Deren Aufgaben sind, die Kommunikation zwischen den unterschiedlichen Fachbereichen zu unterstützen sowie weitere Modelldaten in die übergeordnete Beschreibungsform zu integrieren (z. B. CAD-Daten). Die VIBN-Modelle werden automatisiert auf der Basis der mit CAD-Daten angereicherten Funktionsbeschreibung abgeleitet. Anhand zweier industrieller Fallstudien wurde der Ansatz validiert.

## 4.2 Ausblick

Die Funktionsbeschreibung und das dazugehörige Funktionsmuster dienen als vielversprechende Grundlage für die Etablierung von integrierten, interdisziplinären Modellen. Dadurch lässt sich ein modellbasierter Entwicklungsprozess etablieren, der neben generativen Schritten auch eine Vielzahl an Analysetätigkeiten unterstützen kann.

Folgenden Themenbereiche wurden zukünftig als aussichtsreich identifiziert:

1. Die Definition formaler Anforderungen erlaubt eine präzisere und kompaktere Beschreibung der Anforderungen an eine Maschine oder Anlage. Eine modellbasierte Notation würde die Prüfung von Konsistenzproblemen und die Herstellung von Bezügen zu anderen Entwicklungsartefakten gestatten, die mit einer textuellen Beschreibung nicht möglich wären.
2. Die Generierung von Steuerungscode aus einem Funktionsmodell würde die Entwicklungsgeschwindigkeit erhöhen, da die Funktionen im abstrakten Modell schneller abgebildet werden könnten. Zudem ließe sich die Fehlerbehandlung, die einen wesentlichen Teil des Programmcodes einnimmt, durch eine Angabe von generischen Fehlerbehandlungsroutinen oftmals vollständig ableiten.
3. Das Zusammenwirken mit bestehenden Werkzeugketten umfasst den Datenaustausch mit anderen Entwicklungswerkzeugen, aber auch die teilweise oder vollständige Generierung von Grobmodellen für MCAD- und ECAD-Systeme, für die Mehrkörpersimulation (MKS) oder für eine FE-Simulation.

4. Eine Fehlermöglichkeits- und Einflussanalyse (FMEA) ist eine Analysetechnik zur Vermeidung von Systemfehlern und zur Abschätzung und Eindämmung von deren Auswirkungen, die in vielen Industriebereichen eingesetzt oder sogar vorgeschrieben wird. Durch die Verwendung von geeigneten Modellen ließe sich zum einen der Aufwand für eine FMEA deutlich reduzieren und zum anderen die Genauigkeit der Analyse verbessern.

# Literaturverzeichnis


[BH09] Jewgenij Botaschanjan and Benjamin Hummel. Specifying the worst case - orthogonal modelling of hardware errors. In *Proc. of ISSTA'09*. ACM Press, 2009.

[BHL09] Jewgenij Botaschanjan, Benjamin Hummel, and Alexander Lindworsky. Interdisziplinäre Funktionsmodellierung im Anlagenbau. *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb*, 01-02:71–75, 2009.

[BHLH09] Jewgenij Botaschanjan, Benjamin Hummel, Alexander Lindworsky, and Thomas Hensel. Integrated behavior models for factory automation systems. In *Proc. of ETFA'09*, 2009.

[BHS99] Manfred Broy, Franz Huber, and Bernhard Schätz. AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, 13(13):121–134, 1999.

[BRJ98] Grady Booch, Jim Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[Cui00] Ralf Cuiper. *Durchgängige rechnerunterstützte Planung und Steuerung von automatisierten Montageanlagen*. PhD thesis, Technische Universität München, 2000.

[FWF03] Forschungsvereinigung Werkzeugmaschinen: Richtlinie zur Funktionsbeschreibung, 2003.

[HB08] Benjamin Hummel and Peter Braun. Towards an integrated system model for testing and verification of automation machines. In *MiSE '08: Proceedings of the 2008 international workshop on Models in Software Engineering*, pages 51–56. ACM, 2008.

[Hoa85] Charles A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hum09] Benjamin Hummel. A semantic model for computer-based spatio-temporal systems. In *Proc. of ECBS'09*, 2009.

[ISO94] ISO 10303: Industrial automation systems and integration - Product data representation and exchange, 1994.

[JBLZ09] Benjamin Hummel Jewgenij Botaschanjan, Thomas Hensel, Alexander Lindworsky, and Michael F. Zäh. Simulationsmodelle für die virtuelle Inbetriebnahme. *ATZproduktion*, 05-06:18–22, 2009.

[Pav01] Luis Pavez. *STEP-Datenmodell zur Simulation mechatronischer Systeme. Abschlussbericht des Verbundprojektes MechaSTEP*. Prostep, 2001.

[RJB98] Jim Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[SPHP02] Bernhard Schätz, Alexander Pretschner, Franz Huber, and Jan Philipps. Model-based development of embedded systems. In *OOIS Workshops*, pages 298–312, 2002.

[ZL08] Michael F. Zaeh and Alexander Lindworsky. Virtuelle Inbetriebnahme - Nutzenmaximierung durch Automatisierung der Modellerstellung. In *Internationales Forum Mechatronik - Intelligente mechatronische Systeme*, pages 410–424. Kompetenznetzwerk Mechatronik BW e.V., 2008.

[ZL10] Michael F. Zaeh and Alexander Lindworsky. Automatic model generation for virtual commissioning. In *Proceedings of COMA*, pages 27–32. Dimitri Dimitrov, 2010.