der Bundeswehr
Universität München

Modellierung, Simulation und Transformation von diskreten Prozessen in der Produktion und Logistik auf der Basis von SysML

Dissertation

Oliver Schönherr

Professur für Modellbildung und Simulation

Vollständiger Abdruck der von der Fakultät für Informatik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines Doktors-Ingenieur (Dr.Ing.) genehmigten Dissertation.

Gutachter:

1. Prof. Dr. rer. nat. Oliver Rose
2. Prof. Dr.-Ing. Markus Siegle

Die Dissertation wurde am 27.08.2013 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Informatik am 23.10.2013 angenommen. Die mündliche Prüfung fand am 31.01.2014 statt.

Eingereicht von: Oliver Schönherr

Kontakt: August-Bebel-Straße 15c
01219, Dresden
Oliver.Schonherr@icloud.com

Erstgutachter: Prof. Dr. rer. nat. Oliver Rose
Zweitgutachter: Prof. Dr.-Ing. Markus Siegle

Datum Abgabe: 27.08.2013

Selbständigkeitserklärung

Hiermit versichere ich, die vorliegende Dissertation eigenständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel, angefertigt zu haben. Alle öffentlichen Quellen sind als solche kenntlich gemacht. Die vorliegende Arbeit ist in dieser oder anderer Form zuvor nicht als Prüfungsarbeit zur Begutachtung vorgelegt worden.

Dresden der 27. August 2013,

Oliver Schönherr

Abstrakt

Während sich in vielen Domänen standardisierte Beschreibungssprachen durchgesetzt haben, besteht im Bereich der deskriptiven Modellierung diskreter Prozesse ein Mangel. Darunter ordnen sich beispielsweise Prozessbeschreibungen für die Produktion, Logistik, Krankenhauslogistik oder das Bauwesen unter. Dieser Bereich wird in der Literatur oft auch als Teilbereich der Ereignisdiskrete Modellierung und Simulation eingeordnet. Diese Dissertation beschäftigt sich damit, ein einheitliches Modellierungskonzept für die diskrete Prozessmodellierung bereitzustellen. Mit dem Konzept soll es nicht nur möglich sein, diskrete Szenarien möglichst vielfältig zu modellieren, sondern das Konzept soll auch den Entwurf von dazu passenden Simulatoren unterstützen. Innerhalb der Dissertation wird ein Konzept entwickelt und mit der Modellierungssprache SysML umgesetzt. Das Konzept unterteilt das Gesamtmodell in neun Teilmodelle, um die Komplexität des Gesamten zu reduzieren und die Modellierung möglichst verständlich und mächtig zu gestalten.

In der Praxis werden die beschriebenen Modelle mit einem kommerziellen Simulator abgebildet und simuliert. Um das Modellierungskonzept der Dissertation nutzen zu können, wurde in der Dissertation ein Konzept zur Transformation in Modelle kommerzieller Simulatoren entworfen. Zusätzlich wurden Transformatoren für die Simulationswerkzeuge Anylogic, Flexsim, Simcron und Factory Explorer entwickelt. Um die Gültigkeit der Modelltransformationen testen zu können, wurde zudem ein Konzept für die Verifikation und Validierung der Modelltransformationen erarbeitet.

Um die Modelle möglichst komfortabel erstellen zu können, wurde das Modellierungstool TOPCASED Engineer entwickelt, welches auf dem Open Source Modellierungstool TOPCASED aufbaut. Mit dem Werkzeug können Modelle auf der Basis des entwickelten Konzeptes komfortabel erstellt werden. Um das Modellierungskonzept für die Simulatorenimplementierung zu testen und ein ganzheitlich freies Konzept anzubieten, wurde zudem ein freier Simulator konzipiert.

Abschließend wurde das Modellierungskonzept in den verschiedenen Domänen Halbleiterfertigung, Inlinefertigung von Solarzellen, Montageprozesse, Logistikprozesse und Bauwesen getestet. Alle Domänen konnten ohne größere Schwierigkeiten abgebildet werden. Die jeweiligen Erweiterungen des Metamodelles sind ebenfalls in der Dissertation beschrieben.

Abstract

While standardized description languages are established in many domains there is a deficit in the field of descriptive modeling of discrete processes. This includes for example process descriptions for production, logistics, hospital logistics or the building industry. This field is often referred to as discrete event modeling and simulation in the literature. This dissertation deals with the provision of a homogeneous modeling concept for process modeling. With this concept it shall not only be possible to model discrete scenarios as versatile as possible but it shall also support the design of suitable simulators. In this dissertation a concept is developed and implemented using the modeling language SysML. The concept divides the complete model into nine sub-models to reduce the complexity and to provide a modeling approach which is as coherent and powerful as possible.

In practice the described models are generated and simulated using commercial simulators. In order to use the modeling concept of this dissertation a concept for the transformation into models of commercial simulators was designed. Furthermore converters for the simulation tools Anylogic, Flexsim, Simcron and Factory Explorer were developed. To test the validity of the model transformation a concept for the verification and validation of the model transformation was developed.

In order to be able to create models as comfortable as possible the modeling tool TOPCASED Engineer was implemented which is based on the open source modeling tool TOPCASED. With this tool models based on our concept can be created in a convenient way. To test the modeling concept for the implementation of the simulator and to offer an integrated free of charge concept a gratis simulator was implemented.

Finally the modeling concept was tested in the various domains of semiconductor manufacturing, inline production of solar cells, assembly processes, logistics processes and in the building industry. All domains could be reproduced without major difficulties. The corresponding extensions of the meta model are also outlined in the dissertation.

Vorwort

Zuerst möchte ich mich bei Herrn Prof. Dr. Oliver Rose bedanken, der es mir ermöglichte diese Dissertation zu schreiben. An seinem Lehrstuhl schuf er ein wissenschaftliches Umfeld wie man es sich besser nicht wünschen kann. Zudem stand er mit seinem breiten Fachwissen stets mit Rat und vielen Innovationen zur Seite. Viele Gedankengänge und Konzeptionen sowie die Grundidee dieser Dissertation, gehen von Ihm aus. Weiterhin gilt mein besonderer Dank meinen Eltern Herrn Klaus Schönherr und Frau Ellen Schönherr, die mich in der Zeit der Dissertation immer privat unterstützten.

Zudem konnte ich von den Arbeiten und der Unterstützung vieler großartiger Wissenschaftler profitieren. Herr Tim Weilkiens beschäftigte sich seit der Entstehung von SysML mit der Vermittlung und den Anwendungsmöglichkeiten der Sprache. Er unterstützte die Dissertation mit seinen Arbeiten und seinem persönlichen Rat sehr hilfreich. Herr Leon McGinnis, Professor an der *Stewart School of Industrial & Systems Engineering* am *Georgia Institute of Technology* erarbeitete die ersten Ansätze für die Modellierung von Produktionssystemen mit SysML und begünstigte persönlich mit seinem Rat und Innovationen die Ergebnisse der Dissertation. Auch Herrn Dr.-Ing. Jan Himmelspach und Frau Prof. Dr. Adelinde Uhrmacher der Universität Rostock gilt mein Dank, für die angenehme persönliche Unterstützung bei der Arbeit mit ihren Simulatoren-*Framework* JAMES II.

Weiterhin möchte ich mich bei allen Studenten, die ich im Rahmen dieser Dissertation betreuen durfte, bedanken. Durch ihre Diplom-, Beleg,- und Masterarbeiten leisteten sie einen Beitrag zu den Ergebnissen der Dissertation. Und auch die vielen wissenschaftlichen Mitarbeiter, mit denen ich mich innerhalb und außerhalb des Lehrstuhles von Professor Oliver Rose austauschen konnte, haben zum Endergebnis dieser Arbeit sehr hilfreich beigetragen. Ein letzter großer Dank gilt meinen Korrekturlesern, die ihre Zeit in diese Arbeit investierten.

Inhaltsverzeichnis

1 Einleitung

In vielen Forschungsdisziplinen haben sich Beschreibungssprachen durchgesetzt, darunter die Informatik, die Mathematik oder die Elektrotechnik. Zudem weisen sie eine lange Tradition auf: Die Darstellung der Algebra wurde schon vor ca. 2000 Jahren in dem Werk ‚Arithmetica' des griechischen Mathematikers Dióphantos ho Alexandreús festgehalten (Alten et al., 2003, S. 95 ff.). Die Verwendung von Schaltplänen zur Darstellung elektronischer Schaltungen wurde in den 80er Jahren geprägt und hat sich in Europa nach der Norm der *International Electrotechnical Commission* (IEC) 60617 und im nordamerikanischen Raum nach der Norm des *American National Standards Institute* (ANSI) Y32 als Standard positionieren können (Winzker, 2007, S. 1 f.). In der Informatik hat sich die *Unified Modeling Language* (UML) als standardisierte Sprache zur Spezifikation, Darstellung und Dokumentation objektorientierter Software in den 1990er Jahren durchgesetzt (Skulschus, 2012, S. 131). Im Bereich der Modellierung diskreter Prozesse konnte sich bislang kein Standard etablieren (vgl. Weilkiens, 2008). Doch gibt es in diesem Bereich aus den folgenden Gründen einen deutlichen Bedarf.

1. Durch Beschreibungssprachen können Projekte nach den Prinzipien des *Systems Engineering* umgesetzt werden, um auch bei großen Projekten den Überblick zu wahren und die Unsicherheit der Modelle (Diskrepanz zwischen Modell und Realität) zu reduzieren.

2. Standardisierte Beschreibungssprachen ermöglichen es, sich mit einem einheitlichen Dialekt zu verständigen. Dadurch wird der Austausch und die Verständigung über gemeinsame Vorhaben zwischen verschiedenen Unternehmenseinheiten und Stakeholdern vereinfacht.

3. Beschreibungssprachen sind ein zentrales Element der automatischen Modellgenerierung.

In der Softwaretechnik ist die automatische Codegenerierung von UML-Modellen mittels sogenannter *CASE-Tools* weit verbreitet und standardisiert (Fowler, 2003, S. 23). In der Modellierung diskreter Prozesse hingegen gibt es zwar viele Ansätze, die unter dem Thema „*Model Based Software Engineering*" (MBSE) betrachtet werden (z.B. *Stateflow Coder*, *Advanced Simulation and Control Engineering Tools* (ASCET)), jedoch setzte sich bisher keiner ausreichend durch (vgl. Fachausschuss Software Engineering, 2004). Dies kann durch den Mangel einer einheitlichen, ausreichend mächtigen oder nicht-proprietären Beschreibungssprache bedingt sein. Doch gerade bei der Modellierung diskreter Prozesse im Bereich der Produktion ist die automatische Codegenerierung von Modellen sinnvoll, da in diesen eine Fülle an verschiedenen Werkzeugen eingesetzt wird. Sie sind mangels der Standardisierung oft nicht in der Lage, ihre Modelle auszutauschen.

Die *Object Management Group* (OMG) nahm sich diesem Problem an und veröffentlichte im April 2006 mit der *Systems Modeling Language* (SysML) 1.0 einen Standard, der auf der UML 2 aufbaut

und sich im Bereich des *Systems Engineering* als standardisierte Beschreibungssprache etablieren soll. Seit der Veröffentlichung von SysML, wurde die neue Beschreibungssprache kontrovers diskutiert (vgl. GfSE, 2008). Sie verbreitete sich schnell und wird heute von namhaften Modellierungstool-Herstellern wie ARTiSAN, Telelogic, I-Logix und Sparx Systems unterstützt.

Diese Arbeit versucht, einen allgemeinen Ansatz für die Modellierung von diskreten Systemen aus dem Bereich der Produktion bereitzustellen. Zudem sollen die Ausarbeitungen der Dissertation den Ansatz für die standardisierte Nutzung vorbereiten. Um die Eigenheiten der Modellierung diskreter Modelle zu verstehen, erfolgten einerseits eine umfangreiche Literaturrecherche sowie Expertenbefragung und andererseits zwei weitreichende Marktanalysen von Produktions- und Simulationstools, die in früheren Arbeiten einsehbar sind (vgl. Schönherr, 2008; vgl. Schönherr/Rose, 2009; vgl. Rehm, 2009). Die gewonnenen Erkenntnisse ermöglichen die Erstellung eines allgemeinen Modells für die Modellierung von diskreten Produktionsprozessen, mit dem möglichst umfassend und verständlich verschiedenste Szenarien abgebildet werden können. Zum Umsetzen dieser Aufgabe wird ein umfangreiches Konzept zur Modellierung von Produktionssystemen vorgestellt und gezeigt, wie es sich mit der Modellierungssprache SysML abbilden lässt. Um dem Anwender zu ermöglichen, das Modellierungskonzept frei und möglichst komfortabel zu nutzen, wurde in der Arbeit ein geeignetes SysML-Modellierungswerkzeug entwickelt und bereitgestellt. Die Modellierung der Szenarien dient hauptsächlich deren Simulation und Analyse mit Hilfe eines geeigneten Werkzeuges. Um dies zu ermöglichen, erfolgt die Erarbeitung eines Systems, das es erlaubt, die entstandenen Modelle für verschiedene Simulationswerkzeuge aufzubereiten (zu transformieren). Um die Modelltransformationen verwenden zu können, ist es notwendig, sie zu verifizieren und zu validieren. Für eine vollständig nicht proprietäre Lösung wird ein freies verfügbares Simulationswerkzeug konzipiert. Neben dem Bereich der Produktion erfolgt auch die Betrachtung angrenzender Domänen, die sich mit ähnlichen Problemen beschäftigen.

Vor dem beschriebenen Hintergrund bestehen die Ziele der Arbeit darin,

1. eine Systematik zur Klassifizierung und Beschreibung von Produktionsprozessen zu erarbeiten,
2. ein Konzept zur Abbildung der Systematik mit der Modellierungssprache SysML bereitzustellen,
3. ein Konzept zu evaluieren, mit dem allgemein modellierte Szenarien für eine möglichst breite Menge an Simulationswerkzeugen aufbereitet werden können, und die Entwicklung von entsprechenden Softwarewerkzeugen zur Übersetzung,
4. die Übersetzungen zu verifizieren und validieren,
5. ein geeignetes und freies Werkzeug zum Modellieren der Szenarien zu entwickeln,
6. ein nicht-proprietäres Simulationswerkzeug zu konzipieren,

7. angrenzende Domänen hinsichtlich einer möglichen Adoption des Konzeptes zu betrachten.

1.1 Gliederung der Arbeit

Kapitel 2. Modellieren mit SysML

Das zweite Kapitel stellt die Modellierungssprache SysML vor. Beispiele aus der Produktion sollen dem Leser die verschiedenen SysML-Konzepte erläutern. Es werden nur Teile der SysML erklärt, die für das entworfene Modellierungskonzept relevant sind. Einleitend erfolgt die Betrachtung der Modellierungssprache SysML, ihrer Geschichte und Verwendung. Auch das für diese Arbeit relevante domänenspezifische Modellieren wird genauer betrachtet. Somit vermittelt das zweite Kapitel die notwendigen Grundlagen zum Verständnis der Arbeit.

Kapitel 3. Modellierungskonzept

Das dritte Kapitel stellt den Schwerpunkt dieser Dissertation dar. Dort wird ein Modellierungskonzept entwickelt, mit dem Modelle der Produktion im Untersuchungsgegenstand beschrieben werden können. Einleitend erfolgt ein Überblick über das aus neun Teilmodellen bestehende Gesamtkonzept und seiner Zusammenhänge. Anschließend werden die einzelnen Teilmodelle genau betrachtet. Das Augenmerk der Teilmodelle liegt im Aufbau, den Zusammenhängen und den Modellierungsspezifika. Die Beschreibung des gesamten Modells kann mit SysML erfolgen und soll es ermöglichen, Produktionssysteme detailliert, flexibel und übersichtlich abzubilden.

Kapitel 4. Modelltransformationen

Das vierte Kapitel legt das Augenmerk auf ein Konzept für Modelltransformationen. Die mit dem *Toolkit in Open Source for Critical Applications and Systems Development* (TOPCASED) *Engineer* entwickelten SysML-Modelle sollen für eine breite Menge von Simulationsprogrammen transformiert werden, um sie simulieren und analysieren zu können. Einführend erfolgt die Recherche des wissenschaftlichen Standes der Modelltransformationen und die Erarbeitung eines geeigneten Konzepts. Anschließend werden Transformationen von SysML für die Simulationsprogramme Flexsim, Simcron, Anylogic und Factory Explorer sowie eine Rücktransformation von Flexsim nach SysML aufgezeigt. Besonders interessant sind hierbei semantische Abweichungen der Modellierungskonzepte und der Einfluss auf die Modelltransformationen. Die Modelltransformationen sind notwendig, um mit dem Modellierungskonzept abgebildete Szenarien auch nutzen zu können.

Kapitel 5. Verifikation und Validierung

Im Zentrum des fünften Kapitels steht die Verifikation und Validierung der Modelltransformationen. Dies ist notwendig, um die entwickelten Transformationen bezüglich festgelegter Kriterien auf ihre Korrektheit zu überprüfen. Ohne diese Prüfung könnten die Transformationen nicht genutzt werden, da ihre Korrektheit nur vermutet werden könnte. Einleitend erfolgt dafür eine Betrachtung geeigneter

Techniken im Kontext der Modelltransformation und deren Bewertung hinsichtlich der Anwendbarkeit auf die durchgeführten Modelltransformationen. Die Erarbeitung von Kriterien ermöglicht es, geeignete Methoden zur Verifikation und Validierung der Kriterien zu erörtern. Abschließend wird die praktische Umsetzung der erarbeiteten Methoden anhand einer Modelltransformation veranschaulicht.

Kapitel 6. Das Modellerierungswerkzeug TOPCASED Engineer

Ein passendes Werkzeug für die Ausführung von Modellierungsarbeiten von Ingenieuren sollte möglichst einfach gestaltet sein und keinen proprietären Lizenzeinschränkungen unterliegen. Da kein geeignetes Werkzeug gefunden werden konnte, erfolgte im Rahmen dieser Dissertation die Entwicklung des Modellierungstools TOPCASED *Engineer*. Das entwickelte Werkzeug basiert auf dem *Open-Source*-Projekt TOPCASED, das unter anderem für die Entwicklung von SysML-Modellen zur freien Verfügung steht. Der erste Teil des Kapitels zeigt eine durchgeführte Bestandsaufnahme proprietärer und nicht-proprietärer SysML-Modellierungswerkzeuge. Anschließend wird auf die speziellen *Usability*-Anforderungen eines geeigneten Werkzeuges und deren Umsetzung eingegangen.

Kapitel 7. Entwurf eines Simulationswerkzeuges mit dem Framework JAMES II

Das siebente Kapitel beschäftigt sich mit einem Ansatz für einen eigenimplementierten Simulator. Im Rahmen einer vollständig freien Lösung, muss es möglich sein, Modelle frei zu entwickeln und zu simulieren. Es wurde kein freies Werkzeug gefunden, das ausreichend Möglichkeiten bereitstellt, die vielfältigen Modelle des Konzeptes dieser Dissertation zu simulieren. Daher erfolgte die Entscheidung, einen Simulator zu implementieren. Der Simulator baut auf dem Modellierungskonzept dieser Arbeit auf und wurde mit dem Modellierungsframework *Java based Multipurpose Environment for Simulation II* (JAMES II) der TU Rostock entwickelt. Das Kapitel enthält die Systemstruktur und deren praktische Umsetzung.

Kapitel 8. Domänenspezifische Betrachtungen

Das achte Kapitel untersucht die Eignung des Modellierungskonzeptes für spezielle Domänen aus dem Bereich der Produktion und Domänen anderer Bereiche. Dies ist wichtig, um den praktischen Nutzen des Konzeptes zu überprüfen und zu vergrößern. Es werden die produktionsspezifischen Domänen der Montage und der Halbleiterfertigung sowie die Domänen Logistik und Unikatfertigung im Bauwesen untersucht. Wichtige Kriterien sind die Modellierungsnotwendigkeit, Struktur im Kontext des Untersuchungsgegenstandes und konzeptionelle Modellierungseigenheiten.

Kapitel 9. Abschließende Betrachtungen

Das letzte Kapitel widmet sich den erreichten Ergebnissen der Arbeit, deren kritischer Betrachtung und der Hinterfragung ihrer Zielerreichung. Dabei wird die praktische und wissenschaftliche Eignung der Arbeit eingeschätzt. Abschließend erfolgt die Überprüfung von offengebliebenen Gesichtspunkten, Anknüpfungspunkten und weiterem Forschungsbedarf.

1.2 Einordnung des Untersuchungsgegenstandes

Diese Arbeit untersucht die Modellierung im Bereich der Produktion, deren Ausgangspunkt die optimale Planung zur Herstellung eines Produktes ist. „Mit der Planung herzustellender Produkte, der dafür erforderlichen Produktionsfaktoren sowie der Planung des eigentlichen Produktionsprozesses beschäftigt man sich im Rahmen der Produktionsplanung“ (Domschke/Scholl, 1997, S. 3). Der Untersuchungsgegenstand ist zweidimensional und wird in der Produktion und der Modellbildung abgegrenzt. Erweiternd befasst sich ein Kapitel mit produktionsfremden Domänen, die denselben Restriktionen der Modellbildung unterliegen (vgl. Kapitel 8).

1.2.1 Einordnung des Untersuchungsgegenstandes im Anwendungsbereich der Produktion

Das Buch „Simulation und Optimierung in Produktion und Logistik“ umschreibt die Anwendungen des Untersuchungsgegenstandes gut (März et al., 2011). In der Literatur wird der Bereich der Produktion oft mit dem der Logistik verbunden, unter anderem um dem großen Teil der logistischen Aufgaben in der Produktion gerecht zu werden. In dieser Arbeit wird der betrachtete Bereich statt „Produktion und Logistik“ zur vereinfachten Schreibweise als „Produktion“ bezeichnet. Dies soll nicht zu Missverständnissen bei der folgenden Definition führen.

„Unter Produktion und Logistik werden Fertigungs-, Montage- und Produktionseinrichtungen einschließlich ihrer Prozesse sowie alle Aufgaben der Beschaffungs-, Produktions- und Distributionslogistik verstanden. Die Logistik bezieht sich dabei sowohl auf produzierende Unternehmen als auch auf nicht produzierende Betriebe wie Handelsunternehmen, Flughäfen und Krankenhäuser. Die Abbildungstiefe reicht von der Modellierung übergeordneter Abläufe in Logistiknetzen – beispielsweise auf der Ebene des *Supply Chain Managements* (SCM) – bis hin zur detaillierten Betrachtung einzelner produktions- oder fördertechnischer Abläufe sowie der Anlagensteuerung.

Nicht betrachtet wird hingegen das detaillierte physikalische, kinematische und kinetische Verhalten technischer Systeme. Hierzu zählen beispielsweise urform- oder umformtechnische Prozesse, Schmelzen oder Verformen, Reibungs- oder Kippverhalten sowie Roboterbewegungen. Ergonomiebewegungen unter Verwendung von Menschenmodellen sind ebenfalls nicht Gegenstand der Betrachtung“ (März et al., 2011, S. 6f.).

Beispiele für Anwendungen sind Fertigungsprozesse in der Halbleiterindustrie, Produktionsprozesse in der Schienenherstellung, Montageprozesse bei Anlagenherstellern, Montageprozesse im Flugzeugbau, Produktionsprozesse von Verpackungsanlagen, sequenzielle Produktionsprozesse in der Automobilindustrie, Montageprozesse in der Feinwerktechnik (März et al., 2011). Neben den stark produktionslastigen Prozessen sind auch Anwendungen aus nicht produzierenden Domänen, wie bei Krankenhäusern oder dem Bauwesen sehr ähnlich und werden in der Arbeit in einem gesonderten Kapitel betrachtet.

1.2.2 Einordnung des Untersuchungsgegenstandes im Aufgabenfeld der Produktion

Die Produktionsplanung wird in verschiedene Stufen und Bereiche gegliedert, deren genauere Betrachtung anschließend erfolgt, um den Untersuchungsgegenstand abgrenzen zu können.

Die Stufen der Produktionsplanung werden von verschiedenen Autoren unterschiedlich definiert (vgl. Gutenberg, 1983, S. 97; vgl. Daub, 1994, S. 3; Domschke/Scholl, 1997, S. 8; Kurbel, 1999, S. 116). Diese Arbeit unterteilt die Produktionsplanung in drei Stufen: die Produktionsprogrammplanung, die Bereitstellungsplanung (Mengenplanung) und die Ablaufplanung (Prozessplanung) in Anlehnung an Gutenberg, Daub, Domschke und Scholl (vgl. Gutenberg, 1983, S. 97; vgl. Daub, 1994, S. 3; Domschke/Scholl, 1997, S. 8).

Die Produktionsprogrammplanung legt fest, welche Produktarten in welchen Mengen produziert werden sollen (vgl. Daub, 1994, S. 3; Domschke/Scholl, 1997, S. 8). Ein Unterschied besteht dabei zwischen dem „potentiellen Produktionsprogramm“ einerseits, das strategische (langfristige) Entscheidungen über grundsätzlich zu fertigende Produktarten trifft, und dem „aktuellen Produktionsprogramm“ andererseits, das strategisch und taktisch (mittelfristig) die in einem Zeitraum tatsächlich herzustellenden Produkte nach Art, Menge und zeitlichem Rahmen festlegt (Domschke/Scholl, 1997, S. 9f).

Die Bereitstellungsplanung legt den Bedarf an elementaren Faktoren fest und bestimmt Fertigungslosgrößen (Daub, 1994, S. 3). Hierbei erfolgt die Festlegung strategischer (z.B. Bedarf an Immobilien und maschinellen Anlagen), taktischer (z.B. Bedarf an Maschinen) und operativer Entscheidungen (z.B. Bedarf an Betriebsmittelbestand). Die Materialbedarfsplanung beinhaltet auch das Problem der Losgrößenplanung (Domschke/Scholl, 1997, S. 11f).

„Die Produktionsprozessplanung oder Produktionsdurchführungsplanung beschäftigt sich mit der zeitlichen, mengenmäßigen und räumlichen Planung des Produktionsvollzugs auf Wochen-, Tages- oder Stundenbasis“ (Domschke/Scholl, 1997, S. 15). Auf Grundlage der vorgegebenen Produktionsmengen aus der Produktionsprogrammplanung und Kapazitäten aus der Bereitstellungsplanung werden nun konkrete Fertigungsaufträge gebildet und eingeplant. „Ein Fertigungsauftrag ist eine zeitlich deterministische Arbeitsanweisung zur Herstellung eines Produktes“ (Domschke/Scholl, 1997, S. 15). So wird in der Produktionsprozessplanung beispielsweise entschieden:

- welche Fertigungsaufträge zu bilden sind,
- welche Fertigungsaufträge in welcher Reihenfolge auszuführen sind,
- welche Potentialfaktoren mit welcher Intensität einzusetzen sind und
- welche Verbrauchsfaktoren eingesetzt werden.

Die genannten Punkte sind an detaillierte zeitliche Festlegungen gebunden. (Domschke/Scholl, 1997, S. 15)

Viele Autoren unterteilen die Produktionsplanung in die Bereiche der Ablaufplanung und Ablauforganisation. Im weiteren Sinn ist Ablaufplanung mit der aufgeführten Definition von Produktionsprozessplanung gleichzusetzen (Domschke/Scholl, 1997, S. 29; Daub, 1994, S. 3). Die Ablauforganisation beschränkt sich dem gegenüber nicht nur auf den Produktionsbereich, „sondern befasst sich mit allen Unternehmensbereichen sowie, insbesondere im Rahmen operativer und taktischer Planung, mit Interdependenzen zur Ablauforganisation“ (Domschke/Scholl, 1997, S. 29; Gaitanides, 1983, S. 62).

Neben der Produktionsplanung ist die Produktionssteuerung ein anderer wichtiger Bereich der Produktion, der in diesem Fokus betrachtet werden muss. Die Produktionssteuerung wird meist operativ in Verbindung mit Produktionsplanungs- und Steuerungssystemen (PPS) eingesetzt und unterteilt sich in die Bereiche Auftragsfreigabe, Feinterminierung und Betriebsdatenerfassung (Hansmann, 2006, S. 257). Vor der Auftragsfreigabe wird im Bereich der Betriebsdatenerfassung überprüft, ob die erforderlichen Produktionsmittel vorhanden sind. Wenn bestimmte Ressourcen nicht vorhanden sind, muss umgeplant werden (Domschke/Scholl, 1997, S. 29). Die nun mögliche minutengenaue Eintaktung der Aufträge wird als Feinterminierung bezeichnet (Glaser et al., 1992, S. 183). Dadurch hat die Produktionssteuerung eine überwachende und sichernde Aufgabe.

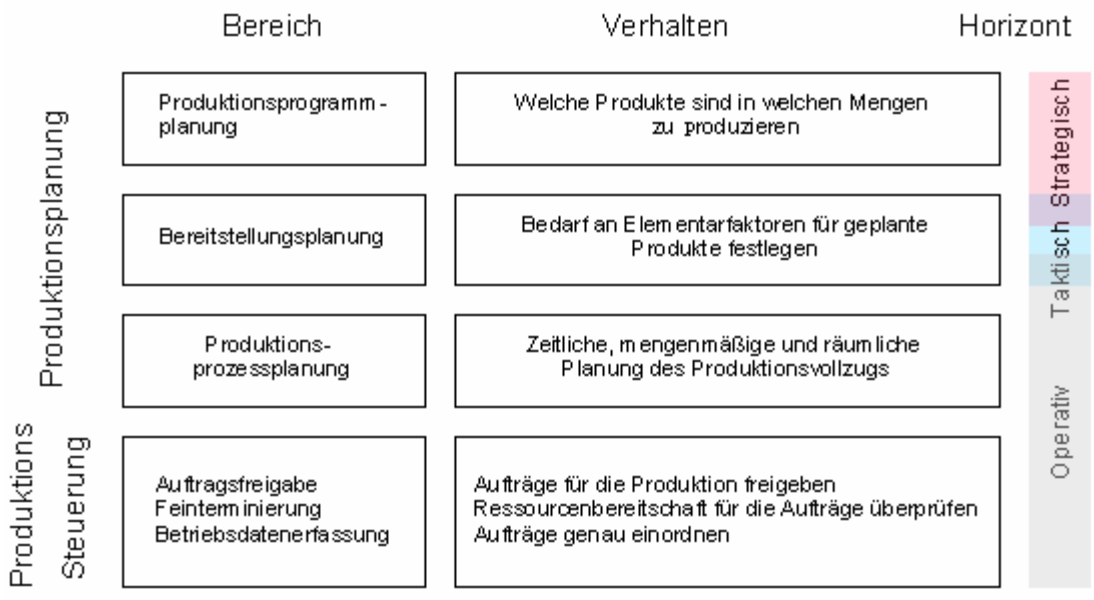


Abbildung 1-1: Systematische Gliederung der Aufgabenfelder der Produktion

Diese Arbeit beschränkt sich auf die operative Planung und somit auf die Betrachtung der Produktionsprozessplanung und Steuerung. Im Folgenden werden also nur Modellierungsnotwendigkeiten dieses Bereichs betrachtet. Wenn von Eigenheiten der Produktion die Rede ist, dann stehen nur jene dieses Bereiches im Mittelpunkt der Betrachtung.

1.2.3 Einordnung des Untersuchungsgegenstandes im Kontext der Modellierung

Zur Bewältigung von Problemen der Produktion werden Modelle genutzt, die als Abstraktion der Wirklichkeit die untersuchte Situation abbilden (Schmidt, 1997, S. 68). Diese können in konstruktive und deskriptive Modelle unterteilt werden, deren Beschreibung aber in der Literatur auch als Optimierungs- und Simulationsmodelle zu finden ist (März et al., 2011, S. 42). Konstruktive Modelle nutzen Restriktionen und Zielkriterien, um eine oder mehrere Lösungen zu erzeugen, „was muss passieren, damit...“. Deskriptive Modelle nutzen Entscheidungen, um das abgebildete System zu untersuchen, „was passiert, wenn…“ (Schmidt, 1997, S. 68). Konstruktive Modelle bieten Algorithmen, mit denen sie eine möglichst optimale Lösung für das Problem berechnen. Um einen Algorithmus anwenden zu können, der das Problem in polynomieller Zeit löst, müssen die konstruktiven Modelle jedoch in ihrer Komplexität und damit auch ihrer Detailgenauigkeit oft sehr stark beschränkt werden. Dadurch kann für ein abstrahiertes Problem oft eine gute oder optimale Lösung gefunden werden, doch ist ungewiss, wie sich diese in der Realität verhält.

Die Erstellung deskriptiver Modelle hingegen erfolgt detailgetreuer. Sie nutzen keinen Algorithmus zur Zielfindung, sondern zeigen anhand von Entscheidungen im modellierten System dessen Reaktion. Dadurch unterliegen sie im Bezug zur Komplexität nicht annähernd der gleichen Begrenzung wie konstruktive Modelle. Der Nachteil deskriptiver Modelle ist, dass die Ermittlung möglichst optimaler Lösungen durch die vielen Details sich eher schwierig gestaltet (Schmidt, 1997, S. 68).

Oft werden beide Modelltypen gekoppelt, um die Nachteile zu relativieren und die Vorteile verbinden zu können (Schmidt, 1997, S. 69; März et al., 2011, S. 42). So ist es beispielsweise möglich die im konstruktiven Modell ermittelten Lösungen im deskriptiven Modell zu evaluieren, bis eine annehmbare Lösung gefunden wurde. Aufgrund des unterschiedlichen Charakters beider Modelle, vor allem unter dem Gesichtspunkt der Komplexität, muss die Modellierung der deskriptiven und die der konstruktiven Modelle separat betrachtet werden. Wegen des stark reduzierten Variablenaufwands ist es beispielsweise möglich konstruktive Modelle als Ansammlung mathematischer Formeln zu betrachten. Daher ist es üblich, diese im *Operational Research* einzusetzen und als Optimierungsmodelle zu bezeichnen. Auch in der Produktion ist es möglich, konstruktive Modelle einzusetzen, indem die Modelle in ihrer Komplexität reduziert werden, um für bestimmte Probleme möglichst optimale Lösungen zu finden (vgl. Law/Kelton, 2000; vgl. Curry/Feldman, 2011). Deskriptive Modelle werden durch den großen Detail- und dadurch Variablenaufwand gern mit graphischen Beschreibungssprachen dargestellt. Hauptsächlich werden diese Modelle durch den beschriebenen Charakter in der Simulation eingesetzt und daher auch als Simulationsmodelle bezeichnet. Diese Arbeit betrachtet deskriptive Modelle. Auch wenn das algorithmische Modell einen konstruktiven Charakter hat, ist es zu abstrakt, um es als konstruktives Modell zu bezeichnen.

1.2.4 Einordnung des Untersuchungsgegenstandes im Kontext deskriptiver Modelle bzw. Simulationsmodelle

Simulationsmodelle werden in der Literatur als statisch oder dynamisch, deterministisch oder stochastisch und (zeit-)kontinuierlich oder (zeit-)diskret klassifiziert (vgl. Law u. Kelton, 2000). Bei statischen Modellen spielt die Zeit keine Rolle, während bei dynamischen Systemen das zeitliche Verhalten das System repräsentiert. Deterministische Systeme enthalten keine Komponenten, die vom Zufall abhängig sind, während stochastische Systeme von zufälligen Ereignissen beeinflusst werden. Kontinuierliche Systeme ordnen jedem reellen Zeitwert ein Zustand des Systems zu. Der Zusammenhang zwischen Ein- und Ausgangsgrößen des Systems sowie dessen inneren Zustand wird meist durch Differentialgleichungen mathematisch beschrieben. Diskrete Systeme ändern sich von einem Zeitschritt zum nächsten, ohne dass zwischen den beiden Zeitpunkten ein Zustandswechsel passiert. Es werden in den folgenden Betrachtungen Modelle mit dynamischen Systemverhalten, stochastischen Komponenten und zu diskreten Zeitpunkten untersucht.

Des Weiteren können deskriptive Modelle (Simulationsmodelle) im herausgearbeiteten Untersuchungsgegenstand ereignisorientiert oder prozessorientiert, sowie aus der Sichtweise der Durchlaufobjekte (*Entiti*es, vgl. Kapitel 3.3) oder der Anlage betrachtet werden. Während bei der ereignisorientierten Sichtweise Ereignisse und Zusammenhänge zwischen Ereignissen explizit dargestellt werden müssen, werden bei der prozessorientierten Sichtweise Prozesse modelliert, die einzelne Aktivitäten wie Arbeitsgänge darstellen. In fast allen kommerziellen Simulatoren wird die prozessorientierte Sichtweise genutzt, sie gilt auch als intuitiver (März, 2011, S. 17). Bei der durchlaufobjektorientierten Sichtweise modelliert der Anwender den Ablaufprozess für jedes einzelne Durchlaufobjekt und beschreibt somit wie sich das Durchlaufobjekt durch das System bewegt. In der Produktion ist das Durchlaufobjekt der Auftrag. Daher wird diese Sichtweise in der Produktion oft als auftragsorientierte Sichtweise bezeichnet. Die anlagenorientierte Sicht modelliert den Anlagenprozess für jeden Prozess explizit und beschreibt das Verhalten des Prozesses beim Auftreten aller relevanten Ereignisse. In kommerziellen Simulationssystemen wird häufig der Ablauf des Durchlaufobjektes explizit modelliert; das grundlegende Anlagenverhalten ist dann oft in vorgegebenen Modellbausteinen enthalten. Diese Umsetzung entspricht der durchlaufobjektorientierten Sichtweise (Fowler/Rose 2004, S. 7 f.). Das vorgestellte Konzept betrachtet die vorwiegend genutzte durchlaufobjektorientierte Sichtweise, von dieser soll auch auf Simulatoren, die die anlagenorientierte Sichtweise vertreten, abgebildet werden.

1.3 Modellierungssprachen im und um den Untersuchungsgegenstand

In der Praxis werden diskrete deskriptive Modelle mit Simulationswerkzeugen modelliert. Diese bauen auf keinem werkzeugübergreifenden Standard auf und sind proprietär, zudem gibt es eine Fülle von

Tools auf dem Markt (vgl. Noche/Wenzel, 2000). Das Fehlen einer standardisierten Lösung zur Erstellung von Modellen im Untersuchungsgegenstand kann daran liegen, dass es keine geeignete Systematik zur Abbildung der Modelle gibt. Im folgenden Abschnitt soll der aktuelle Stand der Modellierung im Bereich des Untersuchungsgegenstands betrachtet werden. Neben im betrachteten Bereich genutzten Modellierungssprachen sollen auch Beschreibungssprachen mit leichten Abweichungen des Untersuchungsgegenstandes betrachtet werden. Eine geeignete Modellierungssprache muss die Kriterien dieser Arbeit erfüllen und somit durch ein Gremium standardisiert, nicht proprietär, mächtig genug zur Beschreibung komplexer Sachverhalte, tauglich für deren (graphische) Darstellung und zur Darstellung von domänenspezifischen Zielräumen geeignet sein.

Im Feld der Modellierung kontinuierlicher Prozesse setzt sich die objektorientierte Beschreibungssprache Modelica (vgl. Modelica, 2013), deren erster Standard 1997 erschien, immer mehr durch. Modelica ist mächtig genug, um beispielsweise Szenarien aus den Gebieten Mechanik, Elektrotechnik, Thermodynamik, Regelungstechnik oder Prozesstechnik abzubilden (vgl. Fritzon, 2011). Die *Modelica Standard Library* (MSL) hält für verschiedenste Fachgebiete freie und quelloffene domänenspezifische Bibliotheken bereit. Es existieren aber auch proprietäre Bibliotheken kommerzieller Anbieter. Zum Entwickeln von Modellen mit Modelica stehen wie bei SysML verschiedene *Open Source*- sowie proprietäre Werkzeuge zur Verfügung. Die Sprache ist heute in der Version 3.3 standardisiert, mächtig in den Darstellungsmöglichkeiten, für die Darstellung komplexer Sachverhalte geeignet und nichtproprietär. Zudem ist sie durch ihr modulares Konzept am domänenspezifischen Problemraum verschiedenster kontinuierlicher Prozesse orientiert (vgl. Fritzson, 2011).

Der erste (Quasi-) Standard, der sich für die Beschreibung diskreter Prozesse verbreitete, waren Petri-Netze (vgl. Petri, 1962). Nachdem sie Anfang der 1980er Jahre in der Informatik an Beachtung gewannen, wurden sie bald sehr vielfältig für die Modellierung diskreter Prozesse wie beispielsweise für die Modellierung von asynchronen Schaltkreisen, Steuerungstechnik oder Geschäftsprozessen eingesetzt (vgl. Reisig, 2010). Selbst ein Teil der UML und SysML (Aktivitätsdiagramm) basiert auf Petri-Netzen (Oestereich, 2006, S. 307.; Störrle, 2005, S. 194). Heute gibt es sie in verschiedenen Ausführungen mit Erweiterungen wie Prioritäten, Zeit, Farben und Attributen, welche jedoch nicht ausreichend standardisiert sind (vgl. Jensen/ Kristensen, 2009). Eine Stärke der Petri-Netze ist, dass Eigenschaften für sie nachgewiesen wurden, die helfen, mit Petri-Netzen dargestellte Sachverhalte zu verifizieren und zu validieren (vgl. Reisig, 2010). Doch werden Petri-Netze bei großen und komplexen Sachverhalten schnell unübersichtlich (vgl. Girault/Rüdiger, 2003). Zwar gibt es auch Arbeiten, die sich mit der Modellierung von Produktionssystemen mit Petri-Netzen beschäftigten, doch beschreiben sie oft weniger komplexe Systeme mit zum Teil nicht-standardisierten Erweiterungen (vgl. Silva/Teruel, 1997; vgl. Zhou/Venkatesh, 1999; vgl. Desrochers/Al-Jaar, 1995). So finden die nichtproprietären und mächtigen Petri-Netze ihre Grenzen in der Eignung für die Darstellung komplexer deskriptiver Modelle und einer mangelnden Standardisierung der erweiterten Varianten.

Viele Autoren sind der Meinung, dass sich sequenzielle Prozesse am besten anhand von Graphen abbilden lassen (vgl. Stachowiak, 1973; vgl. Becker, 1996; vgl. Rupprecht, 2002; vgl. Sauer, 2003). Durch diese Darstellung kann die logische Struktur der Aufgabenstellung mit allen ablauforganisatorischen, technologischen und zeitlichen Informationen bei einer Reihe von Montageablaufstrukturen leicht verständlich und übersichtlich aufgezeigt werden (vgl. Seidel, 1998). Die Abbildung der strukturellen Beziehungen zwischen den Variablen kommt der menschlichen Wahrnehmung und Denkweise nahe. Zudem existieren viele Optimierungsverfahren, die auf Graphen basieren (Majohr, 2008, S. 25). In der Produktion erfolgt die Modellierung von Sachverhalten mit verschiedenen Graphen. Die bekanntesten sind Gozintographen, Prozessketten und Netzpläne. Gozintographen sind gerichtete Graphen und bilden die verschiedenen Herstellungsstufen und die dazugehörigen Zusammenhänge eines Produktes ab (Schneeweiß, 2002, S. 48f; Tempelmeier, 2005, S. 180 ff.). Stücklisten (als tabellarisches Gegenstück zu Gozintographen) werden für verschiedene Aufgabenfelder der Produktion vom operativen bis zum strategischen Bereich genutzt. Durch die tabellarische Darstellung kann die Übersichtlichkeit vielstufiger Produkte schnell verloren gehen (Tempelmeier, 2005, S. 182 f.). Obwohl Gozintographen durch ihre Darstellungsart besser als Stücklisten geeignet sind, um komplexe Sachverhalte darzustellen, eignen sie sich nicht für komplexe deskriptive Modelle.

Netzpläne eignen sich besonders gut zur Aufnahme der Abhängigkeiten von Projektaktivitäten und den dazugehörigen Ressourcen. Durch sie erhält der Betrachter schnell ein rudimentäres Ablauf- und Abhängigkeitsverständnis, ohne dass er genaue Prozesskenntnis besitzen muss (vgl. Pielok, 1995). Sie sind seit langem bekannt und werden in der Praxis häufig verwendet (vgl. Schwarze, 2001). Hier gibt es zwei Varianten die sich durchgesetzt haben: Vorgangsknotennetzpläne (VKN) und Vorgangspfeilnetzpläne (VPN) jedoch sind bei den VKN die Ablaufvarianten des Prozesses unübersichtlich (Majohr 2008, S 30) und bei den VPN in ihrer Mächtigkeit eingeschränkt (Neumann/Schwindt, 1997, S. 205 ff.; Majohr 2008, S 30). Zudem konnte in praktischen Betrachtungen am Lehrstuhl Modellierung und Simulation der TU Dresden festgestellt werden, dass Netzpläne für domänenspezifische Problemräume nicht ausreichend flexibel und anpassbar genug sind.

Am Fraunhofer-Institut für Produktionsanlagen und Konstruktionstechnik in Berlin wurde die zur Darstellung diskreter Prozesse entwickelte, nicht-proprietäre Modellierungssprache MoogoNG entwickelt. Die Sprache stellt die Szenarien als Graphen dar und ist auf das Abbilden diskreter deskriptiver Modelle im Bereich der Geschäftsprozessmodellierung und Produktion ausgerichtet. Das Fraunhofer-Institut stellt für die Modellierung mit MoogoNG ein kostenfreies Werkzeug bereit, das aber noch nicht ausgereift ist. Durch den niedrigen Bekanntheitsgrad der Modellierungssprache kann nicht auf andere Werkzeuge zur Modellierung mit MoogoNG zurückgegriffen werden. Zwar lassen sich einfache Szenarien gut abbilden, doch stellen sich bei komplexeren Szenarien Probleme bei der Darstellungsmächtigkeit und Qualität. Weiterhin fehlen Anpassungsmöglichkeiten bei domänenspezifischen Problemräumen (vgl. Gsuck, 2009).

Wie einleitend erwähnt, gibt es zur Darstellung von diskreten deskriptiven Modellen im betrachteten Anwendungsbereich eine große Menge kommerzieller Werkzeuge, die auch deren Simulation und Analyse unterstützen. Speziell für den Bereich der Produktionsmodellierung haben sich beispielsweise Emplant, Flexsim oder Simcron etabliert, Enterprise Dynamics sowie AutoMod hingegen im Bereich der Logistik (vgl. Noche/Wenzel, 2000). Andere Produkte wie Anylogic oder Arena versuchen, verschiedenste domänenspezifische Problemräume für die diskrete Simulationen abzudecken. In der Praxis hat sich die Verwendung kommerzieller Simulatoren durchgesetzt, doch wird eine Vielzahl von verschiedenen Produkten eingesetzt, was den Austausch in jeder Beziehung erschwert. Standardisierte Modellierungsgrundlagen gibt es in diesem Feld nicht. In dieser Arbeit werden verschiedene kommerzielle Produkte analysiert, um ihre Wissensbasis zu ergründen.

Huang, Ramamurthy und McGinnis zeigten erstmals 2006 eine Möglichkeit, die Struktur eines Produktionssystems mit SysML zu beschreiben. Sie nutzten eine Untermenge der SysML. In ihrer Arbeit erstellten sie für die Struktur eines einfachen Produktionssystems ein Metamodell als Blockdefinitionsdiagramm (Huang et al., 2006, S. 798f). In dieser Arbeit wird dieser interessante Ansatz vertieft betrachtet, um ein komplexes Konzept für die Modellierung in der Produktion und dessen Modellierung mit SysML zu erarbeiten. SysML wird von der OMG standardisiert und ist nicht-proprietär. Nach ausführlichen Tests konnte es als mächtig genug zur Beschreibung komplexer Sachverhalte und tauglich für deren Darstellung befunden werden. Zudem ist SysML sehr gut zur Darstellung domänenspezifischer Zielräume geeignet (Schönherr/Rose, 2009). Sein abstrakter Aufbau erlaubt es SysML, sich an verschiedenen Modellierungskonzepten für Domänenräume zu orientieren. So könnte man mit SysML beispielsweise die verschiedenen Domänenkonzepte aller genannten Simulatoren nachbauen. Hinsichtlich der betrachteten Eigenschaften erfüllt SysML die gestellten Kriterien als Modellierungssprache.

2 Modellieren mit SysML

Das zweite Kapitel gibt einen Überblick über die Anwendung der SysML. Dieser ist für den Leser der Dissertation wichtig, um das vorgestellte Modellierungskonzept - das mit SysML umgesetzt wird – verstehen zu können. Um unnötigen Aufwand zu vermeiden, führt das Kapitel nur Konzepte an, die für das Modellierungskonzept der Dissertation notwendig sind.

Einführend werden Grundlagen von SysML wie der Aufbau, die Entstehung und das Anwendungsgebiet aufgezeigt. Anschließend wird das domänenspezifische Modellieren erklärt, das ein Grundkonzept von SysML darstellt. Der Hauptteil untergliedert sich wie SysML in einen Struktur- und einen Verhaltensteil. Jeweils erfolgt eine Begutachtung der verschiedenen Diagramme auf ihre Eignung in Bezug auf das Modellierungskonzept dieser Dissertation und wie mit den geeigneten Diagrammen modelliert werden kann.

2.1 Grundlagen

In der Softwaretechnik ist die visuelle Beschreibungssprache UML, die von der OMG 1997 zur Spezifikation, Konstruktion und Dokumentation von Software herausgegeben wurde, sehr populär und hat sich dominierend durchgesetzt (Skulschus, 2012, S. 131). In der Vergangenheit gab es etliche Versuche von Systemingenieuren, die UML für ihre Aufgabenfelder zu adaptieren (Hause, 2006, S. 1). Im Jahr 2003 wurde die UML 2.0 von der OMG veröffentlicht, welche zum Entwurf allgemeiner technischer Systeme und nicht nur für Software-Systeme genutzt werden sollte (Hause, 2006, S. 2). Während sich die Spezifikation UML 2.0 in der Softwaretechnik sehr stark verbreitete, konnte sie das im Bereich des *System Engineerings* nicht. Zu stark war das Konzept an das der Softwareentwicklung angepasst (Weilkiens, 2008, S. 24).

Im April 2006 veröffentlichte die OMG und das *International Council on Systems Engineering* (INCOSE) die SysML 1.0, einen Standard, der auf UML 2.0 aufbaut und speziell für die Bedürfnisse von Systemingenieuren angepasst wurde und sich im Bereich des *Systems Engineering* als standardisierte Beschreibungssprache etablieren sollte (vgl. Hause, 2006). Das Ziel der SysML definiert die OMG wie folgt: *„Goal is to provide a standard modelling language for systems engineering to analyse, specify, design and verify complex systems, intended to enhance systems engineering information amongst tools, and help bridge the semantic gap between systems, software and other engineering disciplines“* (OMG, 2010, S. 2).

Im Mai 2011 erschien die SysML-Version 1.3, welche gegenwärtig die aktuelle Version darstellt. SysML wird heute unter anderem in der Industrie und Forschung zur Analyse und zum Design eingesetzt (Incose, 2009). Schon auf dem Tag des *Systems Engineering* 2008 sah man zahlreiche Projekte aus den verschiedensten Anwendungsgebieten, wie beispielsweise Schiffs-, Flugzeug-, oder Kleingerätebau, die mit SysML durchgeführt werden (vgl. GFSE, 2008). Pörnbacher beschreibt, wie er Steuerungssoftware automatisierter Fertigungssysteme mit SysML und Modelica modelliert (Pörnbacher, 2010). Rosenberg und Mancerella entwickeln eingebettete Systeme mit SysML (Rosenberg/Mancerella, 2010). Die Anwendungsgebiete sind sehr vielfältig und erstrecken sich in viele Gebiete des *Systems Engineering*.

Sowohl die UML als auch die SysML sind von der OMG entwickelt worden. Aufgrund dieser Verwandtschaft sind sich beide Konzepte sehr ähnlich. Grundlegend wird in der SysML wie in der UML zwischen Diagramm und Modell, sowie Struktur und Verhalten unterschieden. Das Modell enthält die vollständige Beschreibung des Systems und besteht aus Diagrammen, die einen bestimmten Aspekt visualisieren. Während im strukturellen Modell die Struktur des Systems, also der statische Teil beschrieben wird, stellt das Verhaltensmodell das dynamische Verhalten des Systems dar. Im Vergleich zur UML wurde die Anzahl der verschiedenen Diagrammtypen stark reduziert. Neu sind in SysML das *Parametric*-Diagramm und das *Requirement*-Diagramm. Sowohl die Struktur als auch das Verhalten eines Modells in SysML kann jeweils durch vier Diagrammtypen beschrieben werden (vgl. Abbildung 2-1).

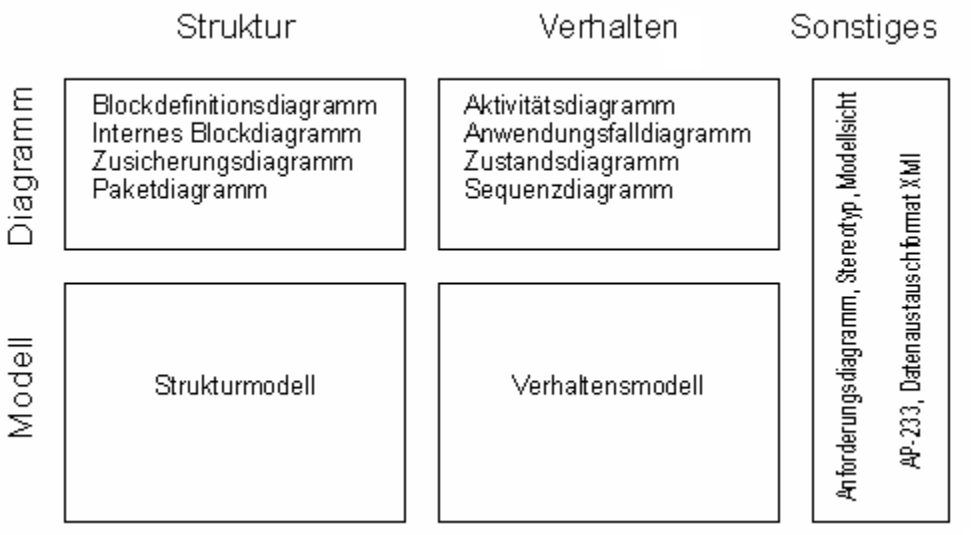


Abbildung 2-1: Der Aufbau von SysML (Weilkiens, 2006, S. 159)

Im Folgenden werden ausschließlich Konzepte tiefergehend erläutert, die für die Umsetzung des Modellierungsansatzes nötig sind. Als Strukturdiagramm wird anlehnend an die Arbeit von Huang, Ramamurthy und McGinnis das Blockdefinitionsdiagramm (vgl. Huang et al., 2007) aber auch das Pa-

ketdiagramm benötigt. Beim Verhaltensmodell wird das Aktivitätsdiagramm, das Zustandsdiagramm und das Sequenzdiagramm genutzt.

Das Anforderungsdiagramm wurde neu in SysML eingeführt. Es ist weder dem Verhalten noch der Struktur zuzuordnen und beschreibt nichtfunktionale Anforderungen, die zu Modellelementen in Beziehung gesetzt werden. Diese Art von Anforderungen spielt hauptsächlich beim Herstellungsprozess neuer Systeme eine überwachende Rolle. Für die Modellierung innerhalb bestehender Systeme ist diese Aufgabe eher untergeordnet und daher nicht Gegenstand näherer Betrachtungen. Die Spezifikation des Anforderungsdiagramms und anderer nicht betrachteter Diagramme wird beispielsweise in Weilkiens (2008) genauer beschrieben. Die anschließenden Erläuterungen gehen hauptsächlich auf SysML-Eigenheiten ein, die Behandlung grundlegender UML-Eigenschaften erfolgt in zahlreichen Lehrbüchern (vgl. Störrle, 2005; vgl. Seemann/Gudenberg, 2000; vgl. Seemann/Gudenberg, 2006). Teilweise werden zur besseren Verständlichkeit Beispiele gegeben, die mit TOPCASED *Engineer* erstellt wurden. Einführend wird das domänenspezifische Modellieren betrachtet, das die Grundlage für die Arbeit mit SysML darstellt. Anschließend wird die Modellierung der Struktur und des Verhaltens in SysML erläutert.

2.2 Domänenspezifisches und Meta-Modellieren

Wer mit einen kommerziellen Simulationsprogramm und der damit verbundenen Modellierungssprache arbeitet, bekommt eine Auswahl festgelegter Elemente und Relationen mit jeweils vordefinierten Eigenschaften und Verhalten (vgl. Noche/Wenzel, 2000; Rehm, 2009, S.67 ff.). Dabei handelt es sich um das Konzept, was wohl der menschlichen Intuition am nächsten liegt. Allerdings bringt es für den Modellierer zwei grundlegende Probleme mit sich. Zum einen ist Semantik und Syntax oft nicht ausreichend dokumentiert, vorhanden oder eindeutig. Zum anderen ist der Nutzer an den gebotenen Umfang an Elementen, Relationen sowie deren möglichen Eigenschaften und Verhalten strikt gebunden (Rehm, 2009, S. 67 ff.; Schönherr, 2008, S. 28 ff.). Das Konzept der Metamodellierung tritt diesen beiden Nachteilen weitestgehend entgegen, da Modelle erweiterbar und Semantik sowie Syntax festgelegt bzw. einsehbar sind. Auch SysML baut auf dem Konzept der Metamodellierung bzw. domänenspezifischen Modellierung auf, was im Folgenden erklärt wird. Zum Einstieg werden die Begriffe Modell und Metamodell im Sinne der Metamodellierung erläutert.

Ein **Metamodell** ist eine Menge von syntaktisch und semantisch definierten nicht konkretisierten (abstrakten) Sprachelementen (Elemente und Relationen mit definierten Eigenschaften). Die Attribute eines nicht konkretisierten Sprachelementes, sind nicht mit Werten spezifiziert. Daher kann es als eine Art Schablone betrachtet werden.

Ein **Modell** ist die Beschreibung eines konkreten Szenarios das aus einer Menge an konkretisierten Sprachelementen eines Metamodells besteht.

In einem Metamodell wird also festgelegt, welche Sprachelemente zur Beschreibung von Sachverhalten zur Verfügung stehen und wie deren Syntax und Semantik definiert ist. In einem Modell erfolgt die Darstellung konkreter Sachverhalte innerhalb der im Metamodell definierten Möglichkeiten. Das Metamodell stellt eine Menge von Sprachelementen als Schablonen bereit, die dann im Modell durch Spezifikation der Elemente ihrer Beziehungen und dazugehöriger Attribute zu Szenarien konkretisiert werden.

Weiterhin besteht die Möglichkeit, Meta-Metamodelle zu definieren. Letztere stellen eine abstrakte Syntax bereit, um Metamodelle zu erzeugen. Die OMG definiert mit der *Meta-Object-Facility* (MOF) ein Meta-Metamodell zur Erzeugung und Spezifikation von plattformunabhängigen Metamodellen (vgl. OMG, 2013). In der MOF wird die abstrakte Syntax zur Beschreibung abstrakter Modellierungssprachen festgelegt, während in aus ihr resultierenden Metasprachen wie der UML oder SysML die abstrakte Syntax und Semantik zum Definieren von Modellen festgelegt wird. Der Metamodellierungsansatz der OMG gibt eine vier-Ebenen Architektur vor, welche für die objektorientierte Arbeit an Software geeignet ist. Für den Ansatz des *Systems Engineering* mit der Nutzung von Stereotypen und aus SysML resultierenden domänenspezifischen Metamodellen ist der Ansatz schwer verständlich. Daher soll er an dieser Stelle nicht weiter ausgeführt werden.

SysML ist eine abstrakte Sprache. Im strukturellen Teil stellt sie Blöcke zur Verfügung, die beliebig benannt und mit Attributen versehen werden können (vgl. Kapitel 2.3.3). SysML definiert die Syntax für die Blöcke, deren Attribute und Relationen, nicht aber deren Semantik. Es ist möglich, einen Block als Ressource zu deklarieren, um ihn mit entsprechenden Attributen zu versehen, aber auch als chemisches Element oder physikalische Einheit mit den entsprechenden Attributen. Um mit SysML in einer bestimmten Domäne zu arbeiten, bietet es sich also an, die abstrakte und in ihren Möglichkeiten sehr freie SysML für sie anzupassen. Um das umsetzen zu können, bietet SysML Stereotypen. Stereotypen erlauben es, einen Block mit Namen, Attributen und Semantik vorzudefinieren oder auch Relationen mit entsprechenden Attributen zu erweitern. Um eine bestimmte Domäne abzubilden, können verschiedene Stereotypen zu einem Profil zusammengefasst werden. Das erstellte SysML-Profil bildet nun das Metamodell, mit dem der Modellierer in der speziellen Domäne arbeiten kann.

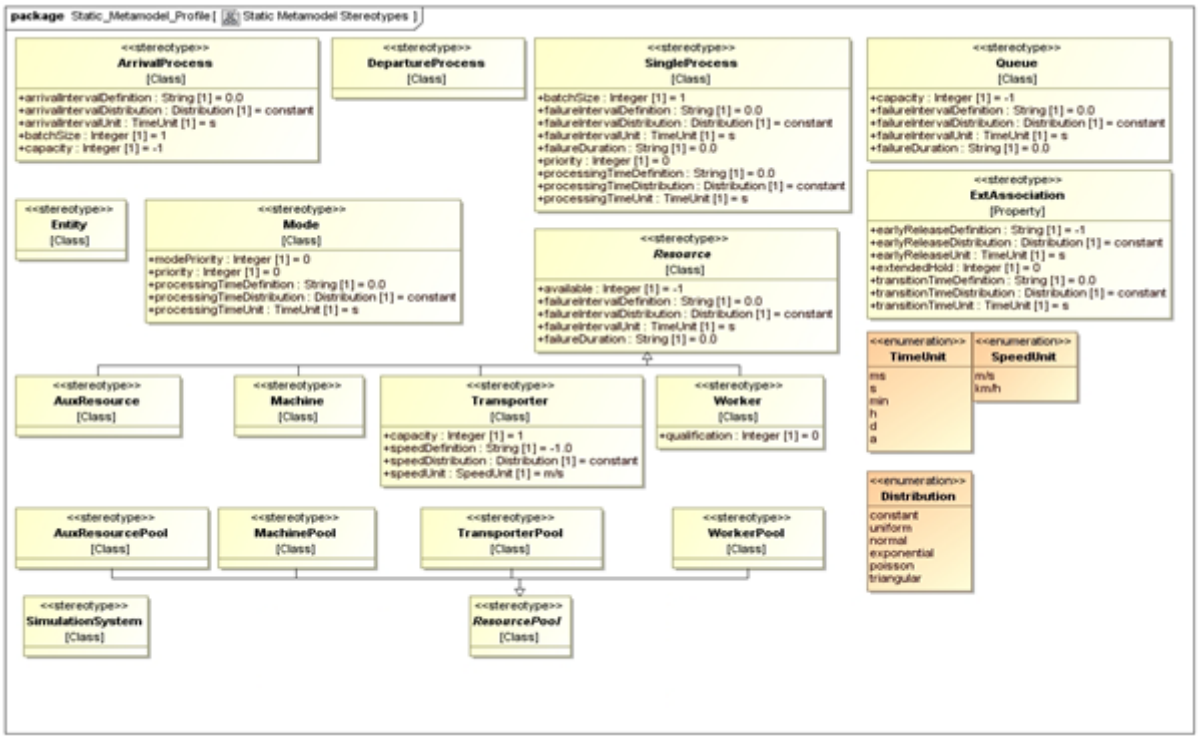


Abbildung 2-2: Metamodell für die Domäne der Produktion

Abbildung 2-2 zeigt ein Metamodell für die Struktur von Produktionsprozessen, welches im Kapitel 3.3 genau erläutert wird. Alle Stereotypen sind im Kopf mit dem Schlüsselwort `<<stereotype>>` gekennzeichnet. Unter dem Namen des Stereotypen steht, welche SysML-Elemente-Form dargestellt wird. Alle in Abbildung 2-2 gezeigten Elemente mit Ausnahme von `ExtAssociation` beziehen sich auf Blöcke. Sie definieren also Schablonen für Elemente. Das Stereotyp `ExtAssociation` bezieht sich auf Eigenschaften von Relationen und ist somit eine Schablone für Beziehungen von Elementen. Oder anders ausgedrückt, weist das Stereotyp `ExtAssociation` den Relationen zwischen den Elementen vom Nutzer definierbare Attribute zu. Enumerationen, die durch das Schlüsselwort `<<enumeration>>` zu erkennen sind, definieren eine Menge von Werten, die dann als Variablentyp für Attribute zur Verfügung steht.

Der Anwender kann mit dem in Abbildung 2-2 beschriebenen Metamodell konkrete Szenarien modellieren. In Abbildung 2-5 ist ein kleines Szenario dargestellt, das auf dem in Abbildung 2-2 gezeigten strukturellen Metamodell aufbaut. Sollte der Anwender Elemente oder Attribute von Elementen oder Relationen benötigen, die im Metamodell nicht vorhanden sind, muss es erweitert werden. Das in Abbildung 2-2 zu sehende strukturelle Metamodell wurde von uns entwickelt, um verschiedenste Szenarien aus dem Bereich der Produktion abbilden zu können. Ähnliche Domänen im Problemfeld wie Krankenhauslogistik, Distributionslogistik oder Bauingenieurwesen lassen sich durch einfache Anpassungen modellieren. Abbildung 2-3 zeigt beispielhaft eine sehr einfache Anpassung des strukturellen Metamodells an die Domäne der Krankenhauslogistik.

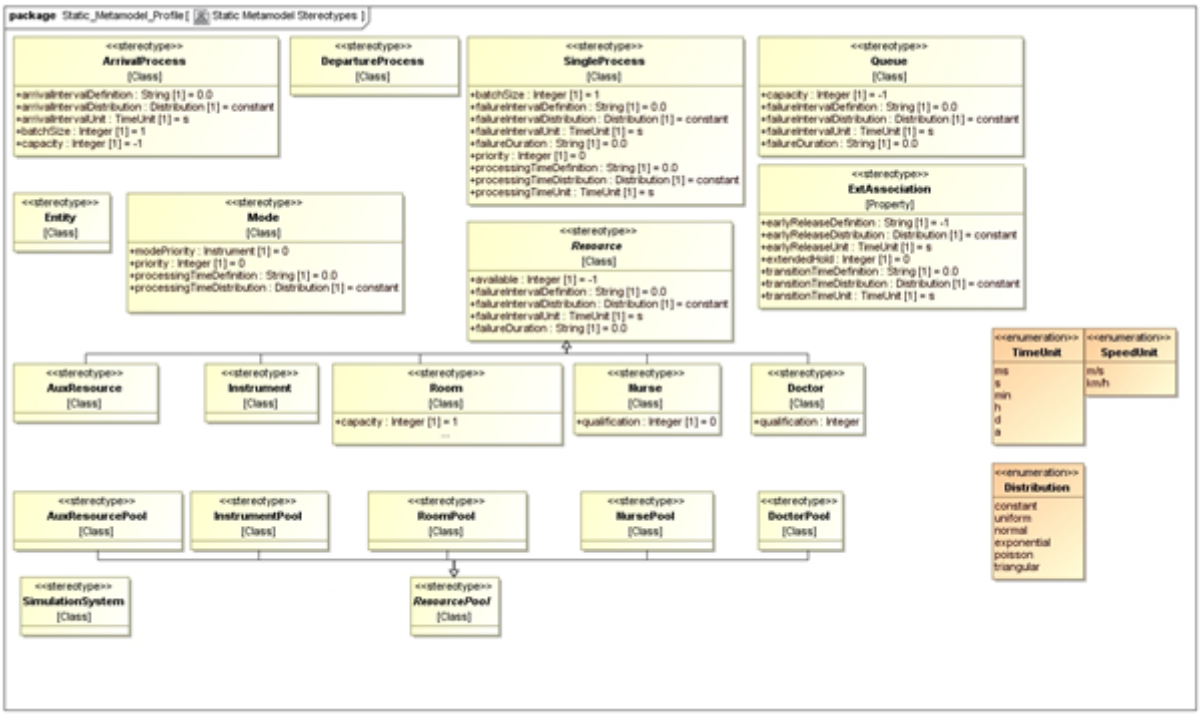


Abbildung 2-3: Metamodell für die Domäne Krankenhauslogistik

2.3 Das strukturelle SysML-Modell

Wie in Abbildung 2-1 gezeigt, kann die Struktur eines SysML-Modells mit Blockdefinitionsdiagrammen, internen Blockdiagrammen, Paketdiagrammen und Zusicherungsdiagrammen beschrieben werden. Das Zusicherungsdiagramm wurde in der SysML neu eingeführt, das Blockdefinitionsdiagramm sowie das interne Blockdiagramm überarbeitet und das Paketdiagram übernommen. Das Objektdiagramm, das Komponentendiagramm, das Profildiagramm und das Verteilungsdiagramm der UML wurden nicht übernommen.

2.3.1 Eignung der SysML-Strukturdiagramme

Im Folgenden soll die Eignung der strukturellen SysML-Diagramme für den im ersten Kapitel abgegrenzten Untersuchungsgegenstand überprüft werden. Das Paketdiagramm wurde aus der UML übernommen. Damit ist es möglich Projekte zu strukturieren, indem die Zuordnung verschiedener Diagramme eines Modells in Paketen erfolgt. Im Modellierungskonzept der Arbeit werden die Teilmodelle zur besseren Übersichtlichkeit einem separaten Paket untergeordnet. So wird beispielsweise das strukturelle Modell mit all seinen Diagrammen oder das Verhaltensmodell mit all seinen Diagrammen jeweils einem separaten Paket zugeordnet.

Das Zusicherungsdiagramm ist ein neues Konzept der SysML. Es beschreibt Zusammenhänge zwischen Eigenschaften verschiedener Blöcke. So lässt sich beispielsweise das physikalische Gesetz von

Newton (Kraft gleich Masse mal Beschleunigung) mit Zusicherungsdiagrammen beschreiben. Die Gesamtmasse des Systems beispielsweise ergibt sich aus der Summe der Masse aller Systembausteine. Die Beschleunigung kann beispielsweise aus Antriebskomponenten eines Flugzeuges berechnet werden. Nun ist die Kraft mit dem Newtonschen Gesetz berechenbar (Weilkiens, 2008, S. 337). In der Produktion wäre beispielsweise Strecke gleich Geschwindigkeit mal Zeit oder Littles Theorem ein Anwendungsfall. Littles Theorem besagt, dass die durchschnittliche Anzahl an Durchlaufobjekten in einen System mit stabilem Zustand, gleich dem Produkt ihrer durchschnittlichen Ankunftsrate und Verweildauer im System ist (vgl. Little, 1961). Es ist möglich Littles Theorem mit dem Zusicherungsdiagramm zu modellieren. Einen solchen übergeordneten Sachverhalt zu definieren bzw. dem Nutzer aufzuzeigen, ist für das vorgestellte Konzept nicht notwendig. Die Umsetzung dieser Zusammenhänge (natürlichen Gesetze) erfolgt im Simulationswerkzeug und ist wie in der realen Welt für das Modell gegeben. Zusicherungsdiagramme werden daher in dieser Arbeit nicht näher betrachtet.

Das Blockdefinitionsdiagramm (BDD) baut auf dem UML Klassendiagramm auf, die Begrifflichkeiten wurden jedoch umbenannt und erweitert. Das BDD ermöglicht es, die Struktur eines Systems zu beschreiben, indem der Modellierer die einzelnen Elemente (Systembausteine bzw. Blöcke), ihre Eigenschaften (Attribute), Hierarchien (Kompositionen, Parts) und Verbindungen (Assoziation bzw. Properties) in einem Blockdiagramm abbilden kann. Anlehnend an die Ausarbeitung von Huang, Ramamurthy und McGinnis wird auch in dieser Ausarbeitung der Großteil der Struktur mit dem BDD abgebildet (vgl. Huang et al., 2007).

Das interne Blockdiagramm in SysML baut auf dem Kompositionsstrukturdiagramm der UML 2.0 auf. Wie beim BDD wurden die Begrifflichkeiten umbenannt und erweitert. Das interne Blockdiagramm beschreibt die interne Struktur eines Blocks, seine Eigenschaften und Konnektoren. „Im internen Blockdiagramm wird dargestellt, wie die Eigenschaften eines Systembausteins miteinander verbunden sind. Es beschreibt also die interne Struktur eines Systembausteins“ (Weilkiens, 2008, S.328). Diese Art von internen Verbindungen ist beispielsweise beim Entwurf technischer Systeme, bei dem SysML vermehrt angewendet wird, von großer Bedeutung (vgl. GFSE, 2008). Für die Modellierung der Struktur von Produktionssystemen im Bereich des Untersuchungsgegenstandes, hat die Modellierung von Verbindungen der Eigenschaften eines Blockes keine Bedeutung. Daher werden interne Blockdiagramme nicht näher betrachtet.

2.3.2 Das Paketdiagramm

Selbst kleinere Probleme haben schnell eine Anzahl von Elementen erreicht, die eine Gruppierung erforderlich macht, um den Überblick wahren zu können. Die Strukturierung mit Paketdiagrammen ähnelt der Struktur von Verzeichnissen auf einer Festplatte. Das für die Gruppierung maßgebliche Kriterium kann nicht expliziert modelliert werden. Das Paketdiagramm bildet die Pakete und ihre Beziehungen ab. Ein Paket bildet einen Namensraum und gruppiert Modellelemente. Alle in einem Paket

enthaltenen Elemente, sind eindeutig über ihren Namen identifizierbar. Daher sind Modellelemente mit gleichem Namen innerhalb eines Paketes nicht erlaubt. Abbildung 2-4 zeigt ein Paketdiagramm. Es gliedert das Modell in Verhaltens- und Strukturmodell. Das Strukturmodell besteht aus den beiden Teilsystemen Fertigung 1 und Fertigung 2. Um beispielsweise auf Elemente des Pakets Fertigung 1 zuzugreifen, definiert SysML die Notation Modell::StrukturellesModell::Fertigung1::Element. Die graphische Abbildung des Paketdiagramms ist an die üblichen Symbole für Verzeichnisse angelehnt.

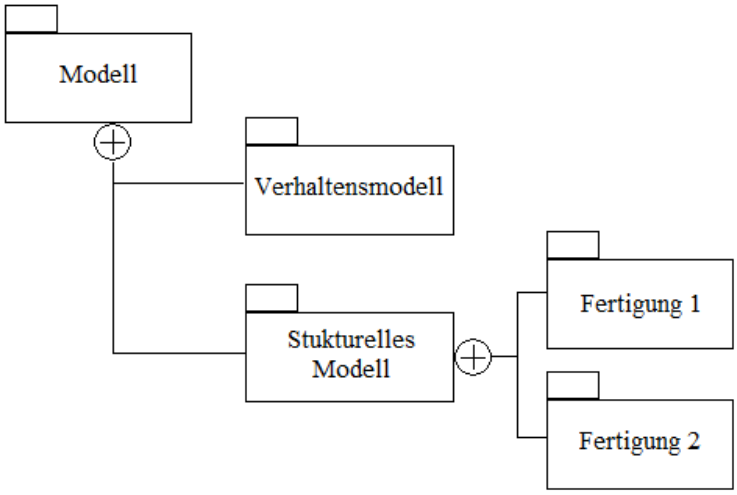


Abbildung 2-4: Beispiel Paketdiagramm

2.3.3 Das Blockdefinitionsdiagramm

In der UML sind die Klassen und deren Objekte die zentralen Elemente. Da diese historisch mit der Softwareentwicklung verknüpft sind, werden sie in SysML bewusst nicht verwendet. Die statischen Konzepte und Gegenstände werden hier als Systembausteine bezeichnet. Es ist wichtig zu beachten, dass dieses Konzept im Vergleich zu UML statt Klassen und Objekten nur Blöcke realisiert. Wie auch Objekte besitzen Systembausteine Attribute (`values`) und Operationen, die sie ausführen können (`operations`). Zusätzlich können sie noch Bedingungen (`constraints`) und Teile, aus denen sie bestehen, ähnlich der Komposition aus der UML, enthalten (`parts`). Diese Strukturmerkmale werden Bausteinattribute genannt und sind alle optional. Für die Modellierung von diskreten Prozessen konnten die `values` alle notwendigen Eigenheiten der Elemente abbilden.

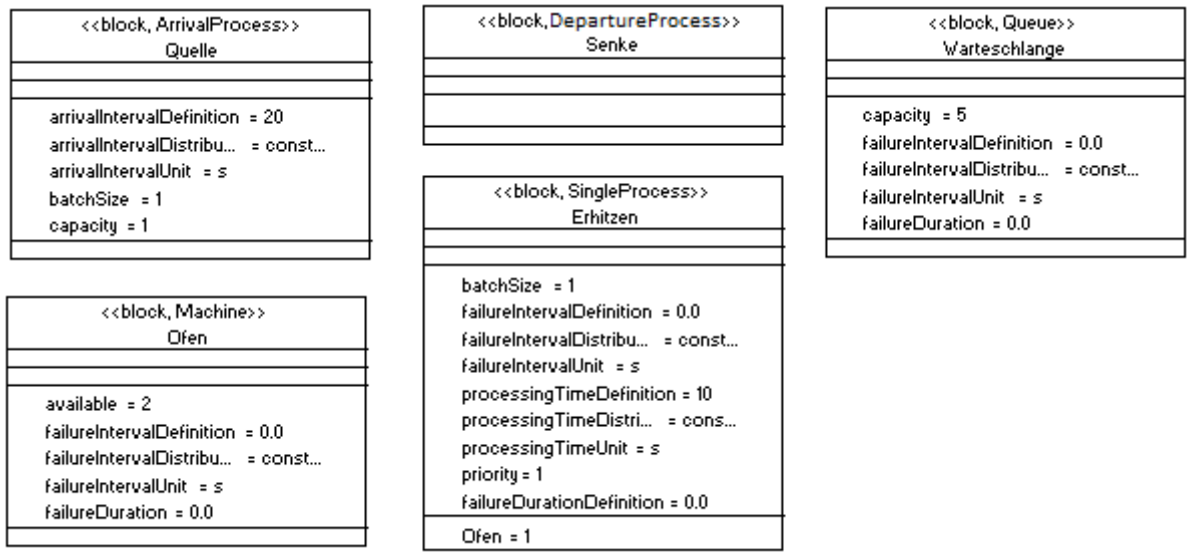


Abbildung 2-5: Aufbau SysML-Systembaustein

Um Referenzen zwischen den verschiedenen Systembausteinen zu schaffen, sind Beziehungen möglich, die als Assoziation oder *Partproperty* bezeichnet werden und Multiplizitäten besitzen können, die Mengenzuordnungen ausdrücken. Es ist möglich Assoziationen wie Blöcke mit Attributen auszustatten, die dann die Beziehung zweier Blöcke mit Eigenschaften belegen. Auch bei der Metamodellierung ist es möglich, die Stereotyp Assoziation mit weiteren vordefinierten Attributen als Stereotyp im Metamodell zu definieren. In Abbildung 2-2 wird die erweiterte Assoziation (`extended Association`) beispielsweise mit zusätzlichen Attributen belegt.

Wie im vorangegangenen Abschnitt erklärt, ist es möglich in der SysML mit Metamodellen zu arbeiten, in denen Elemente und Attribute als Stereotypen vordefiniert sind. Ein Metamodell stellt bei dieser Vorgehensweise Schablonen für die verschiedenen Blöcke bereit (vgl. Kapitel 2.2). Im SysML-Blockdefinitionsdiagramm werden nun verschiedene explizite Ausprägungen der einzelnen Stereotypen modelliert, um ein bestimmtes Szenario auszudrücken.

Abbildung 2-5 zeigt ein kleines Beispiel eines Szenarios der Produktion, das auf dem in Abbildung 2-2 gezeigten Metamodell aufbaut. Es besteht aus einem Ankunftsprozess, einem Abgangsprozess, einem Prozess, einer Warteschlange und einer Maschine. Die Semantik der Attribute ist im Kapitel 3.3 definiert. Das Attribut `available` mit dem Wert `2` von der Maschine `Ofen` bedeutet beispielsweise, dass zwei Maschinen vom Typ `Ofen` im Modell zur Verfügung stehen.

Dem `SingleProcess Erhitzen` ist eine Maschine vom Typ `Ofen` als *Partproperty* zugeordnet. Dies sagt aus, dass der Prozess `Erhitzen` für seine Ausführung ein Element vom Typ `Ofen` benötigt und während seiner Ausführung bindet. Wird die Multipliztität des *Partproperties* erhöht benötigt es mehr Elemente vom Typ der gebundenen *Partproperty*.

2.4 Das SysML-Verhaltensmodell

Die SysML bietet das Aktivitätsdiagramm, das Anwendungsfalldiagramm, das Zustandsdiagramm und das Sequenzdiagramm zum Abbilden des Verhaltens (vgl. Abbildung 2-1). So wurden beispielsweise die anderen Interaktionsdiagramme der UML (Kommunikations-, Zeit-, Interaktionsübersichtsdiagramm) nicht in die SysML aufgenommen. Das Zustandsdiagramm, das Anwendungsfalldiagramm und das Sequenzdiagramm wurden ohne Änderungen aus der UML 2.0 übernommen. Einzig das Aktivitätsdiagramm erfuhr in der SysML eine Anpassung. Die ohnehin schon große Menge an Modellierungselementen der UML 2.0-Aktivitätsdiagramme wurde um Kontrolloperatoren, Raten, Wahrscheinlichkeiten, zeitliche Zusicherungen und spezielle Objekteigenschaften (beispielsweise `non-buffer`, `overwrite` oder `optional`) erweitert. Durch die fehlende Anpassung ist es mit dem Sequenzdiagramm derzeit nicht möglich Objektflüsse, sondern lediglich Operationsaufrufe und Signale, abzubilden (vgl. Weilkiens, 2008, S. 353).

2.4.1 Eignung der SysML-Verhaltensdiagramme

Im Folgenden sollen die einzelnen Verhaltensdiagramme der SysML in Bezug auf ihre Darstellungsmächtigkeit und Eignung zur automatischen Codegenerierung betrachtet werden. Für eine solche Eignung muss die Semantik von Sprachen eindeutig und exakt sein.

Use-Case-Diagramme stammen aus den 70er Jahren und wurden von Ivar Jacobson popularisiert (vgl. Störrle, 2005, S. 150). Sie sind zur Erfassung von funktionalen Anforderungen (was das System ausführen soll) und nicht-funktionalen Anforderungen (alle anderen Randbedingungen) geeignet (vgl. Störrle, 2005, S. 150). Die Semantik des Anwendungsfalldiagramms ist sehr vage definiert und besitzt viele Freiheitsgrade (Weilkiens, 2008, S. 231). Durch die Freiheitsgrade der Semantik sind sie für die automatische Codegenerierung jedoch ungeeignet. Doch werden sie oft als einfach zu verstehen bewertet und dienen daher der Kommunikation von Systemingenieuren und *Stakeholdern*.

Zustandsdiagramme sind hierarchische, endliche Automaten und gehen auf die *ObjectCharts* von Room zurück (vgl. Störrle, 2005, S. 170). Mit ihnen lässt sich das Verhalten von einzelnen Objekten spezifizieren (Seemann/Gudenberg, 2006, S. 105). Sie sind für Objektlebenszyklen, Nutzfälle, Gerätesteuerungen, Protokolle und Dialogabläufe geeignet (vgl. Störrle, 2005, S. 170). Ein Zustandsdiagramm besteht aus Zuständen und deren Zustandsübergängen. „Ein Zustand ist eine Zeitspanne, in der ein Objekt auf ein Ereignis wartet“ (Balzert, 2005, S. 87). Ein Zustandsübergang wird durch ein Ereignis aufgerufen. Ereignisse besitzen keine Dauer, können aber Bedingungen unterliegen. Ein Objekt kann mehrere Zustände durchlaufen. Der nächste Zustand hängt von dem aktuellen Zustand bzw. vom ausgelösten Ereignis ab. Zustände können *entry-, exit-*, und *do-* Aktivitäten besitzen, welche beim Eintreten, Ausführen und Verlassen eines Zustandes ausgeführt werden. Es gibt den Verhaltenszustands- und Protokollzustandsautomaten. Mit dem Verhaltenszustandsautomaten ist es möglich, das

dynamische Verhalten von Klassen und Use-Cases zu modellieren. Der Protokollzustandsautomat drückt aus, in welchem Zustand und unter welchen Bedingungen es möglich ist die Operationen einer Klasse abzurufen (Balzert, 2005, S. 197). Da in der Produktion eher die Abläufe im Vordergrund stehen, werden nachfolgend Verhaltenszustandsautomaten betrachtet.

Sequenzdiagramme, wie auch andere Interaktionsdiagramme, stammen von den Blockschaltbildern der Elektrotechnik ab (vgl. Störrle, 2005, S. 222). Kommunikationsdiagramme sind nicht und Sequenzdiagramme nur theoretisch geeignet, um Verhalten präzise oder vollständig zu definieren (Oestereich, 2006, S. 325). „Sie beschreiben den Ablauf der Kommunikation in einer Gruppe von Objekten“ (Seemann/Gudenberg, 2006, S. 79). „Der Zweck von Sequenzdiagrammen ist es, genau ein Szenario darzustellen und nicht eine Menge von verschiedenen Abläufen. Insofern ist die Möglichkeit mit großer Vorsicht zu genießen. Wenn sie die Vielzahl der Ablaufmöglichkeiten ausdrücken möchten, verwenden Sie besser ein Aktivitätsdiagramm“ (Oestereich, 2006, S. 331). Beim Sequenzdiagramm steht der Verlauf beim Nachrichtenaustausch im Vordergrund. Rollen (Klassen) werden als senkrechte Linien gezeichnet. Die Darstellung der Nachrichten erfolgt waagerecht als Pfeile zwischen den Linien der Rollen. Es gibt synchrone und asynchrone Nachrichten, die optional eine Antwort als gestrichelten Pfeil bekommen können. Es ist auch möglich das Erzeugen und Entfernen von Rollen darzustellen. Ebenso können Zustände notiert werden, in denen sich das Objekt zum jeweiligen Zeitpunkt befindet. Zudem ist es möglich, Sequenzdiagramme verschachtelt darzustellen. Weiterhin existieren verschiedene vordefinierte Operatoren für alternative Abläufe, Verzweigungen und Schleifen.

Aktivitäten haben eine lange Vorgeschichte und gehen auf den Programm-Ablaufplan und das Nassi-Shneiderman-Diagramm zurück (Störrle, 2005, S. 194). Jedoch muss man die Entstehung der UML 1- und UML 2-Aktivitätsdiagramme differenzieren, da sie sich stark in ihrer Ausdrucksweise unterscheiden. UML2-Aktivitätsdiagramme bauen auf der Semantik von Petri-Netzen auf (Oestereich, 2006, S. 307; Störrle, 2005, S. 194). „Aktivitätsdiagramme können benutzt werden, um alle Arten von Abläufen zu beschreiben. Aktivitäten sind sehr ausdrucksmächtig und universell einsetzbar“ (vgl. Störrle, 2005, S. 194). Sie eignen sich besonders für die Beschreibung von Vorgängen, bei denen die Reihenfolge der Schritte eine Rolle spielt, wie der Bearbeitung eines Auftrages in einer Werkstatt (Seemann/Gudenberg, 2006, S. 27 f.). „Mit Aktivitätsdiagrammen ist es möglich, auch sehr komplexe Abläufe mit vielen Ausnahmen, Varianten, Sprüngen und Wiederholungen noch übersichtlich und verständlich darzustellen“ (Oestereich, 2006, S. 303).

Besonders Use-Case-Diagramme, aber auch Sequenzdiagramme sind teilweise vage definiert und daher für die automatische Codegenerierung weniger geeignet. Zustandsdiagramme sind geeignet, um den Übergang der Zustände von Objekten zu spezifizieren. So bietet es sich an, Agenten mit Zustandsdiagrammen zu beschreiben, aber auch wechselnde Zustände von Ressourcen. Die Modellierung des Zustandsmodells des Modellierungskonzeptes erfolgt daher mit Zustandsdiagrammen (vgl. Kapitel 3.7). Für die Beschreibung von Abläufen sind besonders Aktivitätsdiagramme geeignet. Da im Verhal-

tensmodell hauptsächlich Szenarien beschrieben werden, in denen ein Rezept als Ablaufplan dient (vgl. Kapitel 3.4), werden für das Verhaltensmodell Aktivitätsdiagramme genutzt. Im Modellierungskonzept dieser Arbeit gibt es vom Anwender genutzte Diagramme, die transformiert werden. Sie müssen für die automatische Codegenerierung geeignet – also eindeutig definiert – sein. Es gibt aber auch rein informative Konzepte, deren Diagramme nicht für die automatische Codegenerierung geeignet sein müssen (vgl. Kapitel 3.5, Kapitel 3.8, Kapitel 3.9). Für das informative Kommunikationsmodell ist es möglich die zum Nachrichtenaustausch geeigneten Sequenzdiagramme zu nutzen. Für die Use-Case Diagramme finden sich in Modellierungskonzept dieser Dissertation keine Anwendungsmöglichkeiten. Daher ist es nicht notwendig, sie in den folgenden Ausführungen zu betrachten.

2.4.2 Das SysML-Aktivitätsdiagramm

„Das Aktivitätsdiagramm repräsentiert ein Modell, das die Abfolge von elementaren Aktionen beschreibt“ (Weilkiens, 2006, S. 251). Daher ist es für die Beschreibung der Ablaufplanung besonders geeignet. Das SysML-Aktivitätsdiagramm basiert auf dem Aktivitätsdiagramm der UML 2.0, das sich stark von dem der UML 1 unterscheidet. Im Folgenden wird erst das Aktivitätsdiagramm der UML 2.0 beschrieben, wobei nur auf SysML-konforme Eigenschaften eingegangen wird. Anschließend erfolgt eine Erläuterung der Erweiterungen der SysML.

Die Aktivität beschreibt Abläufe und besteht aus mehreren Aktionen. Aktionen sind elementar ausführbare Schritte einer Aktivität. Die Darstellung von Aktivitäten als auch von Aktionen erfolgt als Rechteck mit abgerundeten Ecken. Während der Name einer Aktivität in der oberen rechten Ecke steht, wird der einer Aktion mittig notiert. Innerhalb einer Aktivität wird der Ablauf mittels Knoten und Kanten dargestellt. Zusätzlich ist es möglich, in der oberen rechten Ecke Bedingungen zu deren Ausführung mittels boolescher Ausdrücke zu notieren. Aktivitäten können Eingabe- und Ausgabeparameter enthalten. Diese werden als Rechtecke inmitten des Rahmens gesetzt. Der Name des Parameters ist üblicherweise der des Parametertyps. Aktivitäten werden durch den Aufruf von Eingabeparametern oder Startknoten aktiviert und durch den Aufruf von Ausgabeparametern oder Endknoten beendet (vgl. Abbildung 2-6).

Der Ablauf einer Aktivität wird durch sogenannte Token gesteuert und ist ähnlich dem Ablauf eines Petri-Netzes. Die Ähnlichkeit geht soweit, dass es möglich ist Petri-Netze, als Aktivitätsdiagramme abzubilden (vgl. Störrle, 2004). Des Weiteren lassen sich auch komplexere Aktivitätsdiagramme auf Petri-Netze abbilden (Störrle, 2005, S .194 ff.). Startet eine Aktivität, wird auf jeden ihrer Startknoten ein Kontrolltoken gelegt und auf jeden Eingabeparameter ein Objekttoken. Während Kontrolltoken Informationen beschreiben, stellen Objekttoken wandernde Objekte dar. Objekttoken können von Aktivitäten über Pins als Eingabe konsumiert werden, während Kontrolltoken ohne Pins aufgenommen werden. Innerhalb einer Aktivität ist es möglich, Kontroll- in Objekttoken umzuwandeln und umgekehrt. Nun durchwandern die Token verschiedene über Kanten verbundene Knoten.

Damit ein Token eine Kante überqueren kann, müssen zwei Voraussetzungen gelten:

1. Die an einer Kante notierten Bedingungen müssen erfüllt sein. Diese können boolescher Art sein oder in Form einer Gewichtsangabe vorkommen, die angibt, wie viele Token eine Kante gleichzeitig durchlaufen können.

2. Das Ziel muss bereit sein, einen Token aufzunehmen. Bekommt ein Ziel von zwei Aktionen Token, dann ist dies im Gegensatz zu UML 1 mit einer Und-Verknüpfung verbunden. Es müssen also an beiden Kanten Token bereit stehen, um die Aktion ausführen zu können. Es ist möglich die Und-Verknüpfung in SysML durch das Schlüsselwort <optional>, das an einer der Kanten notiert werden muss, in eine Oder-Verknüpfung umzuwandeln.

Während die Token durch das Aktivitätsdiagramm wandern, können sie auch an Verzweigungen geraten. Hier stößt eine eingehende Kante auf einen so genannten Verzweigungsknoten, aus dem mehrere optionale Abläufe gehen. Die Bedingung zur Wahl einer Kante folgt dem Prinzip „wenn A eintritt, führe B aus, ansonsten C." Ein Verzweigungsknoten wird als Raute notiert, von der beliebig viele Kanten ausgehen. Seine Bedingungen werden über den zugehörigen Kanten in eckigen Klammern notiert (vgl. Abbildung 2-6).

Das Gegenstück eines Verzweigungsknotens ist die Zusammenführung, die ebenfalls als Raute dargestellt wird. Sie bringt mehrere optionale Abläufe zusammen. Das *Splitting* überführt einen Ablauf in mehrere Abläufe. Die Synchronisation führt mehrere nebenläufige Abläufe zu einem zusammen. Beide Elemente werden als schwarzer Balken notiert (vgl. Abbildung 2-6). Wenn sich beim *Splitting* oder bei der Synchronisation mehrere Knoten sammeln (beispielsweise durch die Blockade einer nachfolgenden Aktivität), erfolgt die Weitergabe der Token nach dem *First In First Out* (FIFO)-Prinzip.

Die Knoten und Kanten von Aktivitäten können auf Grund von Gemeinsamkeiten sogenannten Aktivitätspartitionen zugewiesen werden. Damit ist es möglich, dem System Strukturen zuzuordnen. Jede Partition steht für eine Struktur, beispielsweise eine Klasse. Im Bereich der Modellierung von Produktionssystemen ist es günstig, den Partitionen Elemente des Systems zuzuordnen. Abbildung 2-6 zeigt ein Beispiel, in dem ein *Entity* verschiedene Elemente eines Szenarios durchläuft. Das gezeigte Aktivitätsdiagramm ist in Partitionen geteilt, die Elementen der Workstation zugeordnet sind und das Diagramm vertikal trennen.

Durch die Aktion `arrival` treten *Entities* nach der festgelegten Verteilung des Blocks `arivalProcess` in das System ein. Die *Entities* werden zuerst zum `SingleProcess` geschickt und mit der Aktion `working` nach der im Block `SingleProcess` (vgl. Abbildung 2-5) hinterlegten Verteilung bearbeitet. Anschließend wird in einem Entscheidungsknoten ihre Qualität überprüft. Während 95 Prozent der *Entities* an den Endprozess weitergeleitet werden und über die Aktion `departure` das System verlassen, müssen fünf Prozent an der Warteschlange überprüft werden. Aus der Warteschlange wandern 40 Prozent der *Entities* wieder zum `SingleProcess` und der Rest kann das System durch den `DepartureProcess` verlassen.

Auch SysML erweitert die UML 2.0-Aktivitätsdiagramme um einige Komponenten, die für die Modellierung von Produktionssystemen wichtig sind. So ist es möglich, Aktivitätskanten mittels Raten eine zeitliche Frequenz zuzuordnen, in der Elemente ein- oder ausfließen (vgl. Abbildung 2-6). Die Rate kann diskret oder kontinuierlich sein und wird in geschweiften Klammern über der Kante gesetzt (beispielsweise <discrete{rate=1/Minute}>).

Auch die Entscheidungen wurden in der SysML erweitert und können statt booleschen Bedingungen auch anhand von Wahrscheinlichkeiten die Token-Nutzung regulieren. Um Wahrscheinlichkeiten auszudrücken, wird über den ausgehenden Kanten einer Verzweigung das Schlüsselwort `probability` gefolgt von einem passenden Wert notiert (vgl. Abbildung 1-5). Letztlich wurde in SysML der schon erläuterte Typ `optional` eingeführt, mit dem Aktivitäten, die mehrere Eingänge besitzen, starten können, ohne dass alle mit Token belegt sein müssen.

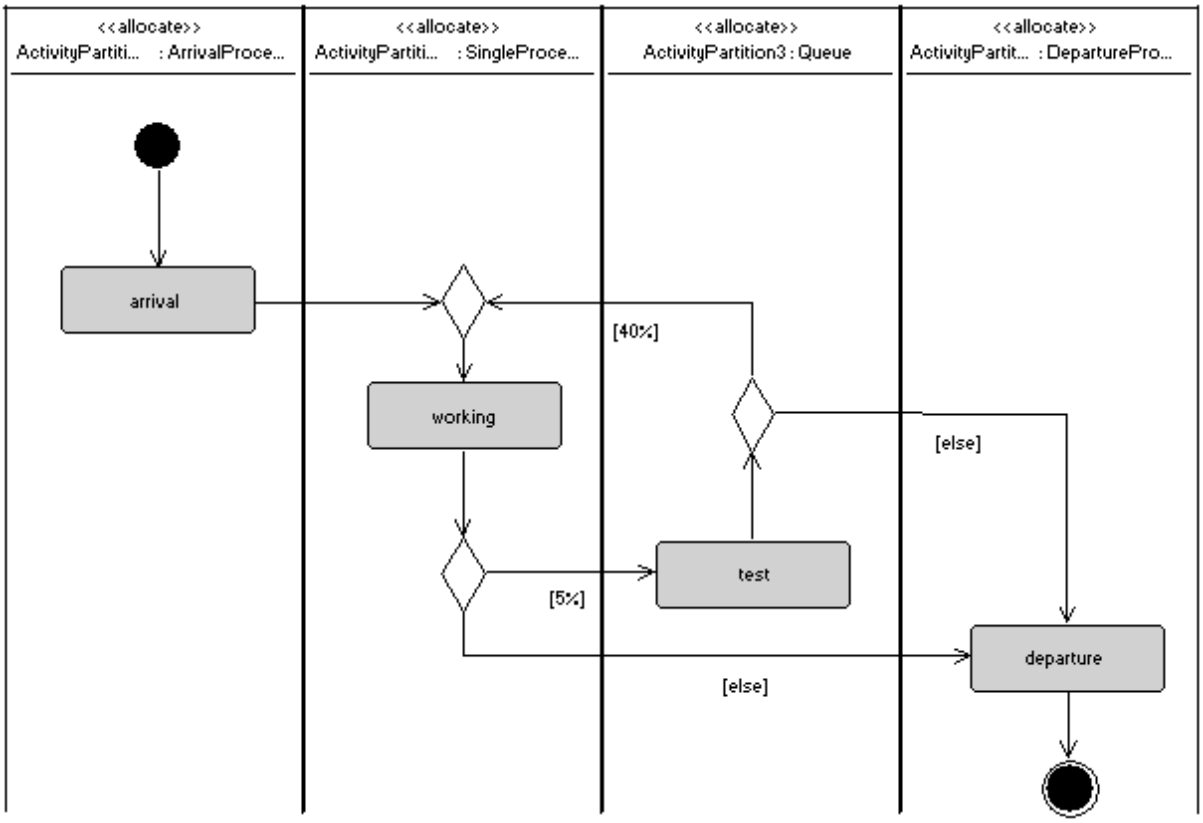


Abbildung 2-6: Beispiel Aktivitätsdiagramm

Wie beschrieben, werden zur Modellierung der Ausführungsebene Aktionen genutzt. Die UML 2.0, wie auch die SysML, stellt ca. 40 vordefinierte Aktivitäten (vgl. Spezifikation UML; vgl. Spezifikation SysML). Die wichtigsten für diese Arbeit sind `CellBehaviorAction`, `TimeAction`, `AcceptEventAction`, `SendSignalAction`, `ReceiveSignalAction`, `readStruc-`

turalFeature und addStructuralFeatureValue. Weitere Aktivitäten werden in verschiedenen Büchern ausführlich besprochen (vgl. Weilkiens, 2006 b; Rumbaugh et. al., 2004).

Die CallBehaviorAction wird wie die übliche Aktivität mit einem Rechteck mit abgerundeten Ecken notiert, enthält aber zusätzlich noch ein Forkensymbol (vgl. Abbildung 2-7). Diese Aktion ruft ein Verhalten ab, das eine Interaktion, ein Zustandsautomat oder wieder eine Aktivität sein kann. Da in dieser Arbeit nur Aktivitätsdiagramme für das Verhaltensmodell genutzt werden, wird der Aufruf von CallBehaviorAction auf Aktivitäten beschränkt.

Das Senden und Empfangen von Signalen erfolgt über die SendSignalAction und AcceptEventAction. Die SendSignalAction wird als eine Art Pfeil notiert, der das Senden von Signalen symbolisiert. Die AcceptEventAction wird durch ein Rechteck mit eingehendem Pfeil symbolisiert (vgl. Abbildung 2-7). Die ReceiveSignalAction empfängt ein Zeitsignal wie beispielsweise eine Zeitüberschreitung.

Die TimeAction ruft als Signal nach einer festgelegten Zeit (*rate*) eine Aktion auf. Sie wird als eine Art Sanduhr abgebildet (vgl. Abbildung 1-6). Alle anderen Aktionen wie auch addStructuralFeatureValue werden als Rechteck mit abgerundeten Ecken dargestellt. Die Aktion addStructuralFeatureValue kann Attribute setzen, während readStructuralFeature Atribute lesen kann (vgl. Schönherr, 2008).

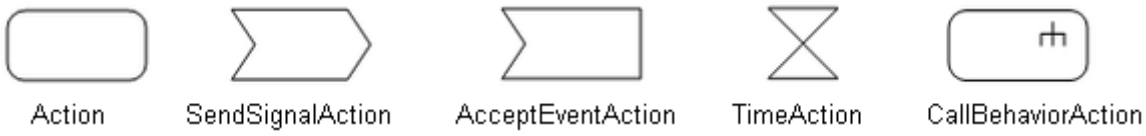


Abbildung 2-7: SysML Aktionen

2.4.3 Das SysML-Zustandsdiagramm

Ein System befindet sich immer in einem Zustand, der aus Teilzuständen besteht und dem Kombination von Werten zugrunde liegt. Ereignisse, die auf ein System treffen, können zu Änderungen des Zustands führen. Es ist möglich, diesen Sachverhalt mit dem SysML-Zustandsdiagramm zu beschreiben. Es besteht aus Zustandsautomaten mit Zuständen und deren Zustandsübergängen, die von Ereignissen ausgelöst werden. „Der Zustandsautomat beschreibt die möglichen Zustände und Zustandsübergänge einer Struktur“ (Weilkiens, 2008, S. 262). Das Kontextobjekt eines Zustandsautomaten ist in der Regel ein Block, dessen Verhalten beschrieben wird. Der Zustandsautomat besitzt keine eigene Notation und wird durch das komplette Zustandsdiagramm ausgedrückt. Die Beziehung des Zustandsautomaten und

des zugehörigen Blocks hat kein graphisches Element, ist aber explizit im Modell vorhanden. Der Zustandsautomat besteht aus Zuständen, Pseudozuständen, Transitionen und Regeln zur Ausführung. Ereignisse kommen in die implizit vorhandene Warteschlange des Automaten und werden nach der Reihenfolge ihres Auftretens von Transitionen und Zuständen konsumiert. Erst wenn alle Aktionen eines Ereignisses abgeschlossen wurden, wird das nächste Ereignis ausgeführt. Jedoch muss das Zustandsverhalten nicht beendet sein, um das nächste Ereignis zu starten. Ereignisse, die nicht konsumiert werden können, da beispielsweise keine passenden Transitionen vorliegen, werden verworfen.

Ein Zustand repräsentiert eine Menge von Werten eines Blocks. So könnte beispielsweise der Zustand `arbeitsbereit` eines Prozesses für eine Wertekombination vorhandener Ressourcen und eines *Entities* stehen. Zustände werden durch Rechtecke mit abgerundeten Klammern symbolisiert. Im Zustand können interne Verhaltensweisen stehen, die wie Transitionen durch Verhaltensweisen eintreten, aber nicht zum Beenden des Zustands führen müssen. Zustände können `entry-, exit-,` und `do-` Aktivitäten besitzen, welche bei dem Eintreten, Ausführen und Verlassen eines Zustandes ausgeführt werden. Als Erweiterung der UML können Zustände auch in einem ihm untergeordneten Zustandsdiagramm genauer beschrieben werden (Kapselung). Es können auch mehrere Zustände gleichzeitig in einem System bestehen. Dazu gibt es die Abbildung mit orthogonalen Zuständen (Weilkiens, 2008, S.265f.).

„Die Transition spezifiziert einen Zustandsübergang. Sie ist eine gerichtete Beziehung zwischen zwei Zuständen und definiert Auslöser und eine Bedingung, die zum Zustandsübergang führen, sowie ein Verhalten, das während des Übergangs ausgeführt wird“ (Weilkiens, 2008, S. 266). Es können Bedingungen zur Ausführung von Transitionen definiert werden. Eine Transition wird durch einen durchgezogenen gerichteten Pfeil symbolisiert und erhält die Beschriftung `Auslöser1, Auslöser2, … [Bedingung] / Verhalten`. Es ist möglich, die Bedingung in jeder beliebigen Sprache (beispielsweise in der *Object Constraint Language* (OCL) oder Programmiersprache) als booleschen Ausdruck anzugeben. Besonders bei orthogonalen Transitionen hängen die Zustände von den Zuständen anderer Regionen ab. So kann eine Bedingung auch prüfen, ob Zustände anderer Regionen aktiv sind. Eine alternative Notationsweise, in der Bedingungen in Signalnotation und Aktionen in Rechtecken notiert werden, wird hier nicht besprochen (Weilkiens, 2008, S. 267 f.).

„Ein Auslöser referenziert genau ein Ereignis und stellt die Beziehung zu einem Verhalten her. Ein Ereignis spezifiziert das Auftreten einer Erscheinung, die bzgl. Ort und Zeit messbar ist“ (Weilkiens, 2008, S. 268). Ein Auslöser an einer Transition notiert, aktiviert die Transition bei Eintritt des Ereignisses. Auslöser sind das Bindeglied zwischen Ereignissen und Verhalten in einem Modell. Es gibt vier Arten von Ereignissen:

1. das Aufrufereignis, bei Aufruf einer Aktion (beispielsweise `reserviereRessource()`),

2. das Änderungsereignis, bei Erfüllung eines booleschen Ausdrucks, wenn sich im System ein Wert geändert hat (beispielsweise `[Ressource.verfügbar() r > 10]`),
3. das Signalereignis, bei Empfangen eines Signals (`Ressource(kommt)`) und
4. das Zeitereignis, beim Auftreten eines Zeitpunktes `at(<zeit>)` (beispielsweise `at(ankunft))` oder nach bestimmter Zeit `after(<zeit>)` (beispielsweise `after (2 Minuten)`).

Zudem haben Zustandsautomaten auch Pseudozustände, eine Transition einen Start- und mehrere End-Zustände, die wie im Aktivitätsdiagramm notiert werden. Der Startzustand ist kein echter Zustand und springt daher sofort auf den ersten Zustand. Ist keine weitere Region des Zustandsautomaten aktiv, beendet der Endzustand den Automaten. Weitere Pseudozustände sind Splitting, Synchronisation und Entscheidungen, die wie bei Aktivitätsdiagrammen Eingangs- und Ausgangspunkte sowie eine tiefe oder flache Historisierung darstellen können (vgl. Weilkiens, 2008, S. 271ff.).

Abbildung 2-8 zeigt beispielhaft ein Zustandsdiagramm für das Element Ressource. Der Startzustand ist ein Pseudozustand und wird beim Starten des Systems aktiviert. Die Ressource kann dann durch das Eintreffen von Ereignissen, die an den Transitionen notiert sind, in Zustände übergehen. Beispiele für Ereignisse sind in Abbildung 2-8 das Binden oder Freigeben eines Prozesses, oder das Eintreten der Eigenschaft `failure`, eines Fehlers.

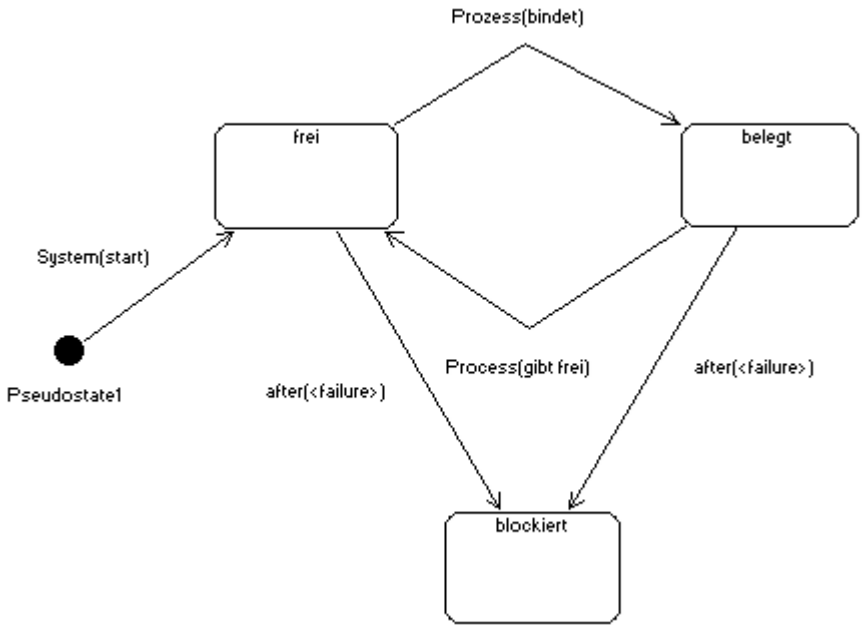


Abbildung 2-8: Beispiel Zustandsdiagramm

2.4.4 Das SysML-Sequenzdiagramm

Das Sequenzdiagramm ist das einzige Kommunikationsdiagramm, das die SysML aus der UML übernommen hat. Es beantwortet die Frage: „Wann ruft wer wen wie auf?“. Im Gegensatz dazu beantwortet das Aktivitätsmodell die Frage: „Was passiert in welcher Reihenfolge?“ (Weilkiens, 2008, S. 275). „Die Interaktion beschreibt eine Kommunikation zwischen Lebenslinien. Die Kommunikation basiert auf dem Austausch von Nachrichten in Form von Operationsaufrufen oder Signalen“ (Weilkiens, 2008, S. 276). Kommunikationspartner werden durch Lebenslinien repräsentiert, die Kommunikation durch Nachrichten. Ein Diagramm beschreibt eine Interaktion und hat einen konkreten Ablauf. Es wird modelliert, wann wer wem eine Nachricht zusendet. Im Kopf der Lebenslinie wird der Name des zugeordneten Modellelements notiert. Auf der Lebenslinie werden mit Balken die Stellen gegenzeichnet, an denen das zugehörige Element aktiv ist.

Sowohl der Aufruf einer Operation als auch die Übertragung einer Nachricht werden durch eine durchgezogene Linie vom Sender zum Empfänger charakterisiert. Die Antwort erfolgt immer durch einen gestrichelten Pfeil mit geschlossener Pfeilspitze. Nachrichten können synchron oder asynchron verlaufen. Synchrone Nachrichten werden durch eine ausgefüllte Pfeilspitze abgebildet und warten auf eine Antwort des Empfängers, bis die Lebenslinie des zugeordneten Objekts weiter verfolgt wird. Asynchrone Nachrichten werden durch eine offene Pfeilspitze symbolisiert und warten nicht auf eine Antwort. Objektaufrufe werden durch einen gestrichelten Pfeil mit offener Pfeilspitze vom Sender zum Empfänger gekennzeichnet.

Des Weiteren stehen dem Sequenzdiagramm sogenannte kombinierte Fragmente zur Verfügung, die es ermöglichen, bedingte Ausführungen eines Operanden, parallele Ausführungen, Schleifen und andere Ablaufvarianten zu modellieren. Ein kombiniertes Fragment wird durch ein Rechteck mit durchgezogenen Linien und einem kleinen Pentagon in der oberen rechten Ecke notiert. In dem Pentagon ist der Typ des Interaktionsoperators notiert. Gestrichelte Linien trennen die einzelnen Teile des kombinierten Fragments. Bedingungen für die einzelnen Fragmente stehen in der Nähe der ersten Lebenslinie. Insgesamt gibt es zwölf Interaktionsoperatoren, Verzweigungen und Schleifen (`alt, opt, break, loop`), Filterungen und Zusicherungen (`critical, neg, assert, consider, ignore`) sowie Nebenläufigkeit und Ordnung (`seq, strict, par operator`). Die genaue Erklärung aller Operanden kann in der UML 2.0- Zertifizierung eingesehen werden (vgl. Weilkiens/Oestereich, 2006). Beispielsweise beschreibt `alt` einen optionalen Ablauf, `if, then, else, break` eine bedingte Schleife und `par` parallele Prozesse.

Mit Interaktionsreferenzen ist es möglich, Sequenzdiagramme wie Funktionen beim imperativen Programmieren wiederaufzurufen. Die Interaktionsreferenzen werden wie kombinierte Fragmente mit dem Interaktionsoperator `ref` symbolisiert. Der im Fragment notierte Name wird als Bezeichner des verwendeten Sequenzdiagrammes verwendet und wird durch dieses im Ablauf substituiert.

Weiterhin sind auch Bedingungen direkt in Lebenslinien möglich, die als Zustandsvarianten bezeichnet werden (Weilkiens, 2008, S. 286). Auch zeitliche Zusicherungen sind möglich, die auf das einfache *SimpleTimeModel* der UML aufbauen und auch bei Aktivitätsdiagrammen angewendet werden können (Weilkiens, 2008, S. 287). Weitere Möglichkeiten zur Modellierung zeitlicher Aspekte bietet das *UML-based Design Framework for Time-triggered Applications* (vgl. Nguyen et al., 2007; vgl. Weilkiens, 2008, S. 275 ff.).

Abbildung 2-9 zeigt ein Beispiel mit den für diese Arbeit wichtigsten Eigenschaften des Sequenzdiagrammes. Der Prozess möchte eine Ressource reservieren. Um dies zu erreichen, schickt er eine Anfrage an den Kontroller. Der Kontroller ist nun aktiv. Der zweite aufgesetzte Balken symbolisiert die reservierende Aktivität des Kontrollers. Erst führt er Berechnungen durch, um dann die Ressource durch einen synchronen Aufruf zu reservieren. Dadurch, dass auch der Aufruf des Prozesses synchron war, wartet er, bis ihm die Ressource vom Kontroller abschließend zugesprochen wird.

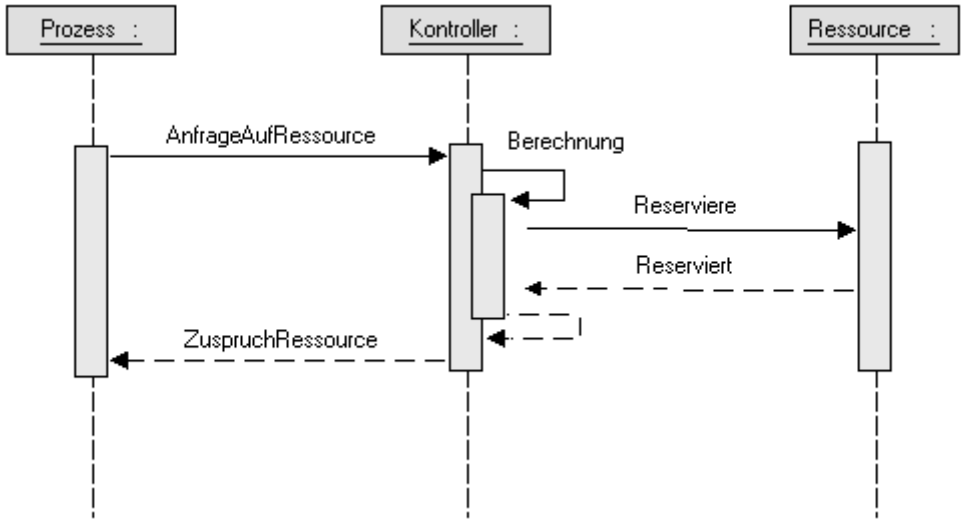


Abbildung 2-9: Beispiel Sequenzdiagramm

3 Konzeption der Modellierung

Auf der Basis konzeptioneller Forschungsarbeiten (vgl. Schönherr/Rose, 2009) wurde ein allgemeines Modellierungskonzept zum Abbilden von Produktionssystemen im Untersuchungsgegenstand entworfen. Um die Eigenheiten der Modellierung in der Produktion zu verstehen, wurden neben einer umfangreichen Literaturrecherche und Expertenbefragung zwei weitreichende Marktanalysen von Produktions- und Simulationswerkzeugen durchgeführt (vgl. Schönherr/Rose, 2009; vgl. Rehm, 2009). Jeder Simulator hat eine spezielle Art, Sachverhalte abzubilden und versucht, die Modellierung von Sachverhalten umfassend abzudecken und komfortabel zu gestalten. Diese Modellierungserkenntnisse der Hersteller wurden genutzt, um die Modellierung von Sachverhalten weitgehend zu verstehen und strukturieren zu können. Besonders geeignet ist der Ansatz dieser Arbeit für die Domäne der Produktion und für die Modellierungssprache SysML. In der Literatur konnte keine ähnliche Gesamtkonzeption der Modellierung im Untersuchungsgegenstand gefunden werden. Zum Herleiten einzelner Teilmodelle war es jedoch möglich, Fachliteratur zu nutzen.

Wichtig war es, das Konzept strukturiert, flexibel, erweiterbar und vollkommen offengelegt zu gestalten. In herkömmlichen Simulatoren wird das Modell in der Regel nur schwach nach semantischen Teilstrukturen getrennt. Teilweise gibt es sogar keine Unterscheidung zwischen Simulator und Modell (vgl. Abbildung 3-2). Die Sichtweise des Anwenders auf das Modell ist daher weniger strukturiert. Weiterhin erschweren die in sich gebundenen Konzepte eine Erweiterung des Modellierungskonzepts durch den Entwickler. Die Ziele der Strukturierung und Erweiterbarkeit des Konzeptes sind also komplementär zueinander. Um das Modellierungskonzept dieser Arbeit möglichst strukturiert zu gestalten, wird es in neun Teilmodelle aufgespalten (vgl. Abbildung 3-2). Das komplette Modell und somit alle Teilmodelle sollen vollständig vom Anwender eingesehen werden können. Das soll den Anwender darin unterstützen, die Prozesse optimieren zu können.

Da die Modelle in der Produktion durch ihre Komplexität und ihr Domänenspektrum sehr vielfältig sein können (März et al., 2011, S. 6 f.), ist es wichtig, das Konzept so zu gestalten, dass der Anwender sehr flexibel modellieren kann. Jeffrey stellt die Anforderung Flexibilität für das Modellieren von adaptierbaren Produktionsmodellen und verweist vor allem auf Aktions-, Zustands-, *Entity*- und Maschinen-Flexibilität (Jeffrey et al., 2000, S. 4 f.). Auch das Ziel der Flexibilität ist mit dem der Strukturiertheit komplementär. Umso differenzierter die einzelnen Teilaspekte eines Modells betrachtet werden, desto flexibler ist es möglich innerhalb dieser Teilaspekte zu modellieren. Um Flexibilität und Erweiterbarkeit zu gewähren, wurde zudem darauf geachtet, die Schnittstellen der Teilmodelle klar zu gestalten, die semantischen Elemente in jedem Modell disjunkt und die Anzahl gemeinsamer Restriktionen durch die Modellierung zweier Sachverhalte in verschiedenen Teilmodellen gering zu halten.

In den folgenden Ausführungen wird zuerst die Gesamtkonzeption der Modellierung betrachtet, um anschließend vertiefend auf die einzelnen Teilmodelle einzugehen. Abschließend erfolgt eine umfassende Betrachtung der Gesamtkonzeption mit den gewonnenen Spezifika der untersuchten Teilmodelle. Neben den Ausführungen des Modellierungskonzeptes wird parallel die Nutzung des Konzeptes mit der SysML erläutert.

Das Konzept kann von einem Ingenieur als Anwender genutzt werden, um Simulationsmodelle aus der Produktion abzubilden. Anschließend ist es möglich diese mit Hilfe eines Transformators für ein bestimmtes Werkzeug (beispielsweise Simulator, *Scheduler*) zu transformieren. Zum Erstellen eines Transformators wird ein Entwickler benötigt. Das Konzept kann aber auch von einem Entwickler genutzt werden, um Simulatoren konzeptionell zu entwerfen und anschließend zu implementieren. In den folgenden Ausführungen des Modellierungskonzeptes werden daher die beiden Rollen des Anwenders und des Entwicklers unterschieden. Auf bestimmten Teilen des Modells kann nur der Entwickler arbeiten (beispielsweise das Zustandsmodell oder das Kontrollmodell). Doch kann der Anwender die Funktionsweise dieser Modellteile durch SysML-Diagramme einsehen.

3.1 Ereignisgesteuerte diskrete Simulatoren

Um die Zusammenhänge der folgenden Ausführungen besser verstehen zu können, ist es wichtig zu wissen, wie der Simulator auf dem aufgezeigten Modell arbeitet. Daher wird einleitend die Funktionsweise eines für den Untersuchungsgegenstand geeigneten Simulators erklärt. Dieser Typ von Simulatoren wird in der Literatur als *Discrete Event Simulator* (DES) bezeichnet. Die folgenden Ausführungen über DES sind an die Arbeiten von März et al (2011, S. 12 ff.) und Law/Kelton (2000) angelehnt.

Zu Beginn werden die Zustandsvariablen des Modells, die Statistikvariablen, der Ereigniskalender und die Simulationsuhr initialisiert. Dann wird ein Zyklus durchgeführt, in dem die Zeitführungsroutine das zeitlich am nächsten liegende Ereignis vom Kalender holt, die Simulationsuhr auf den Zeitpunkt des Eintrittes dieses Ereignisses gestellt wird und die zugehörige Ereignisprozedur aufgerufen wird. Innerhalb der Ereignisprozedur werden die Zustandsvariablen abgeändert, die Statistikvariablen angepasst und ggf. Folgeereignisse im Ereigniskalender eingefügt. Abschließend wird geprüft, ob die Simulation weiter ausgeführt werden soll. Wenn ja, beginnt der Zyklus erneut, sonst erfolgt eine Auswertung der Statistikvariablen, um mit ihnen ein Bericht zu erstellen. Der Simulator springt also von einem Ereigniszeitpunkt zum nächsten und ändert dabei den Systemzustand (vgl. Abbildung 3-1).

„Der Kern jeder Simulationssoftware, die nach dem DES-Prinzip arbeitet, besteht aus den folgenden Komponenten (Law u. Kelton 2000):

- Systemzustand: Menge von Variablen, die das System zu einem bestimmten Zeitpunkt beschreibt,
- Simulationsuhr: Variable mit dem augenblicklichen Wert der simulierten Zeit,
- Ereigniskalender (Ereignisliste): Liste mit Zeitpunkten des nächsten Eintritts jedes Ereignistyps (ggf. mit Parametern für diesen Ereignistyp),
- Statistische Zähler: Variablen zur Speicherung von statistischen Informationen über das Systemverhalten.
- Initialisierungsroutine: Unterprogramm zur Initialisierung der Variablen am Beginn der Simulation,
- Zeitführungsroutine: Unterprogramm zur Bestimmung des nächsten Ereignisses und zum Vorstellen der Simulationsuhr auf den nächsten Ereigniszeitpunkt,
- Ereignisroutine: Unterprogramm, das nach Simulationsende aus den statischen Zählern Schätzwerte für gewünschte Systemleistungsgrößen berechnet und in Form eines Berichtes ausgibt,
- Hauptprogramm: Unterprogramm, das durch die Zeitführungsroutine das nächste Ereignis bestimmt und die entsprechende Ereignisroutine aufruft. “ (März et al., 2011, S. 14 f.)

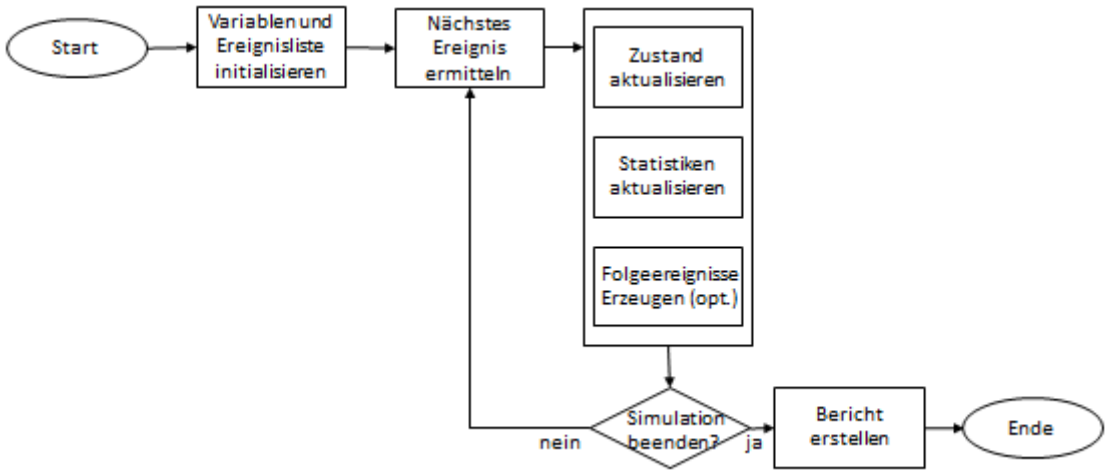


Abbildung 3-1: Algorithmus DES (vgl. März et al., 2011, S. 14)

Nach diesem einfachen Algorithmus ist jede DES-Software implementiert. Mit einem solchen Softwaresystem „lassen sich Experimente mit beliebiger Form von ereignisorientierten Systemmodellen durchführen“ (März et al., 2011, S. 16). Dies zeigt, welche große Rolle das Modell spielt und welchen Umfang es beschreibt. Natürlich ist es auch möglich unsaubere Implementierungen zu generieren, in der das Modell und der Simulator vermischt werden. Das soll im Konzept dieser Arbeit grundlegend ausgeschlossen werden. Teile des Modells sind in den meisten Simulatoren nicht vom Nutzer zu be-

einflussen (beispielsweise das Verhalten in gewissen Detaillierungsgraden, Zustände, Kommunikationsabläufe). Der Anwender arbeitet also immer innerhalb eines vom Entwickler vorgegebenen Metamodells, das der Simulator ausführen kann. In manchen angebotenen Simulatoren sind Teile des Modells nicht für den Anwender einsehbar (beispielsweise das Kommunikationsmodell oder das Kontrollmodell). Das kann den Eindruck bei den Anwendern erwecken, dass Teile des Modells Bestandteile des Simulators sind, was nicht wünschenswert ist.

3.2 Überblick des Modellierungskonzeptes

Abbildung 3-2 zeigt auf der linken Seite ein Beispiel für ein herkömmliches Modellierungskonzept, wie es in vielen Modellierungswerkzeugen vorkommt. Auf der rechten Seite ist das Modellierungskonzept dieser Dissertation dargestellt. In dem Standard-Modellierungskonzept werden die einzelnen Teilmodelle beim Großteil der Simulatoren ineinander vermengt. Oft wird nicht einmal zwischen Simulator und Modell unterschieden, und Teilmodelle sind nicht für den Anwender des Werkzeuges einzusehen. Wegen der gewünschten klaren Strukturierung wird in dieser Arbeit grundsätzlich zwischen Modell und Simulator unterschieden. Des Weiteren wird versucht, die semantischen Teilstrukturen als unterschiedliche Aspekte des Modells zu beleuchten und in unterschiedlichen Teilmodellen zu differenzieren.

Das Systemmodell besteht aus sieben Teilmodellen: Drei davon kann der Anwender bearbeiten, die übrigen vier dienen ausschließlich seiner Information. Das Systemmodell zeigt alle für den Anwender relevanten Informationen der Fertigung auf. Zudem legt es sämtliche Elemente, Verhaltensweisen, Steuerungsmechanismen, Zustände, Ereignisse und Kommunikationsabläufe fest. Ferner werden das experimentelle Model und das Analysemodell außerhalb des Systems definiert. Beide legen fest, wie der Simulator auf dem Systemmodell arbeiten soll und welche Simulationsergebnisse (Statistiken) dem Anwender zurückgegeben werden.

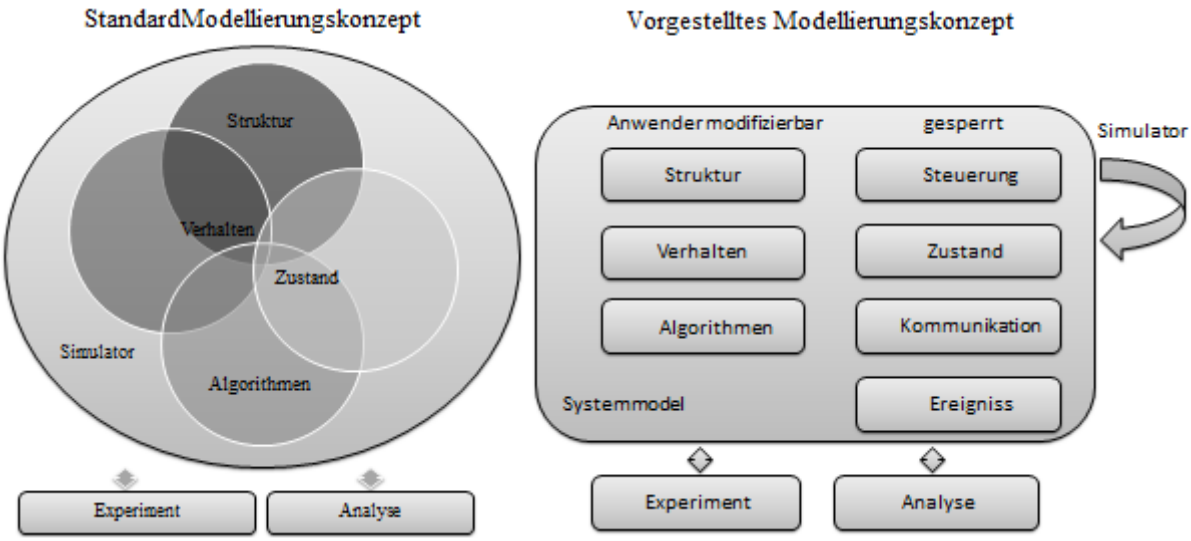


Abbildung 3-2: Gegenüberstellung des vorgestellten Modellierungskonzeptes mit dem Standardkonzept

Das strukturelle Modell beschreibt die statische Struktur des Systems, wie die Elemente, deren Attribute sowie deren Beziehungen. Beispielsweise sind Ressourcen, Prozesse und deren Beziehungen Teile des statischen Modells. Die SysML bietet zum Beschreiben der Struktur eines Modells vier Diagramme, von denen in diesem Konzept das Paketdiagramm und das Blockdefinitionsdiagramm genutzt werden (vgl. Kapitel 2.3).

Das Verhaltensmodell beschreibt das dynamische Verhalten der Elemente bzw. zwischen ihnen, beispielsweise das Wandern des Werkstückes durch die Produktionsanlage einer Fertigung. Das Verhaltensmodell kann in verschiedenen Granularitätsstufen bzw. Detaillierungsgraden beschrieben werden. Zur Abbildung des Verhaltens wird das SysML-Aktivitätsdiagramm genutzt (vgl. Kapitel 2.4).

Das Zustandsmodell beschreibt, welche Verhaltenszustände die Elemente der Simulation annehmen können. Jedes Element des Modells hat ein eigenes Zustandsmodell. Dieses Modell ist für die Elemente eines Stereotypen jeweils gleich (beispielsweise für alle Prozesse). Um die Zustandsmodelle zu beschreiben, werden SysML-Zustandsdiagramme verwendet. Zustände, Ereignisse und das Verhaltensmodells sind voneinander abhängig.

Das strukturelle Modell, das Verhaltensmodell und das Zustandsmodell bilden die Grundstruktur eines Systems ab und werden im Folgenden als Basismodell bezeichnet (vgl. Abbildung 3-31).

Die algorithmischen Entscheidungen können den vollständigen Systemzustand einbeziehen. Alle Elemente des Systems sind mit dem Steuermodell verbunden, das den kompletten Systemzustand kennt. Wenn eine Entscheidung innerhalb eines Systemelements getroffen werden soll, führt das Steuermodell den Algorithmus aus und gibt das Ergebnis / die Entscheidung zum betreffenden Element zurück. Der Anwender kann verschiedenste Algorithmen im algorithmischen Modell definieren, die das Steuermodell dann ausführt. Das Steuermodell definiert spezielle Elemente (beispielsweise Monitor, Kon-

troller) sowie Schnittstellen zwischen dem Steuermodell, dem algorithmischen Modell und dem Basismodell. Der Anwender kann diese Informationen zwar einsehen, doch die Implementierung erfolgt ausschließlich durch den Entwickler. Das Kontrollmodell wird mit dem SysML-Zustandsdiagramm beschrieben.

Das algorithmische Modell bietet dem Nutzer eine stereotypisierte Auswahl von Algorithmen an. Daraus kann er verschiedene Algorithmen wählen und bestimmen, über welche Teile des strukturellen Modells die Algorithmen in welcher Art und Weise (d.h., mit welchen Startwerten) arbeiten sollen. Für jeden Algorithmus kann der Anwender bestimmte Startwerte festlegen – beispielsweise, welche Anzahl an Elementen notwendig ist, um in einer *Constant Work in Process* (CONWIP) Regelstrecke neue Elemente einzuschleusen . Die gewählten Algorithmen werden dann über die zugeordneten Elemente vom Kontroller berechnet (vgl. Kapitel3.6.1). Neben den vorgefertigten Algorithmen, kann der Anwender auch neue Algorithmen aus Einzelteilen anderer Algorithmen zusammensetzen. Das algorithmische Modell wird mit Hilfe von SysML-Blockdefinitionsdiagrammen und SysML-Verhaltensdiagrammen beschrieben.

Das Kommunikationsmodell ist rein informativ für den Anwender. In ihm werden die Kommunikationsabläufe des Systems mit Sequenzdiagrammen festgelegt. Die Kommunikationsabläufe im Untersuchungsgegenstand begrenzen sich auf den Austausch zwischen Basis- und Steuerungsmodell.

Im System gibt es Aufrufereignisse, Änderungsereignisse, Signalereignisse und Zeitereignisse (vgl. Kapitel 2.4.3). Sie gehen mit Verhaltensphasen des Systems einher und können zu Zustandsübergängen führen. Zudem sind sie die Schnittstelle zwischen dem Systemmodell und dem Simulator (vgl. Kapitel 3.1).

Genauso wie das Analysemodell ist das experimentelle Modell kein Teil des Systemmodells. In diesen Modellen wird festgelegt, wie der Simulator auf dem Systemmodell arbeiten soll und welche Simulationsergebnisse der Anwender zurückbekommt. Im experimentellen Modell definiert der Nutzer unter welchen experimentellen Aspekten die Simulation ablaufen soll (beispielsweise der Simulationszeitraum). Das experimentelle Modell kann mit stereotypisierten Blöcken im Blockdefinitionsdiagramm dargestellt werden. Im Analysemodell bestimmt der Anwender, welche Faktoren (Statistiken) betrachtet werden sollen (vgl. Abbildung 3-2; vgl. Abbildung 3-32).

3.3 Das strukturelle Modell

Im ersten Abschnitt zum strukturellen Modells wird dessen Modellierung gegliedert. Im zweiten Abschnitt wird ein grundlegendes strukturelles Metamodell für Produktionsprozesse gezeigt. Das Wort „grundlegend“ bedeutet, dass das strukturelle Metamodell eine Menge an Produktionsprozessen abbilden kann und trotzdem nach den einleitend genannten Grundsätzen flexibel und erweiterbar bleibt.

Um das strukturelle Metamodell nach diesen Grundsätzen zu entwickeln, wurde besonders eine der beiden einführend erwähnten Untersuchungen von Simulationswerkzeugen genutzt. In der Studie wurden die Simulationswerkzeuge Simul8, Process Simulator, AnyLogic, Simcron Modeller, Flexsim, und Plant Simulation betrachtet (vgl. Schönherr, 2008). Die Tools sind alle im Kapitel 3.5.1 beschrieben, in der Rehms Marktuntersuchung diskutiert wird. Beide Untersuchungen betrachten fast dieselben Simulationswerkzeuge, jedoch unter anderen Aspekten. Während Rehm (2009) die Werkzeuge auf ihre Steuerung hin untersucht, betrachten Schönherr und Rose (2009) die Simulatoren hinsichtlich deren Strukturierung.

3.3.1 Grundlegende Betrachtungen

Das strukturelle Modell beschreibt, welche Objekte sich in einem System befinden und welche Beziehungen zwischen diesen Objekten bestehen. Es können die drei Klassen imaginäre, reale und Hilfsobjekte, unterschieden werden (vgl. Abbildung 3-3). Die imaginären Objekte, Ankunft- und Abgangsprozess, Warteschlange und Prozess bzw. Prozessschritt, werden für den Ablauf der Simulation benötigt und sind Teil vieler Forschungskonzepte, wie beispielsweise der Warteschlangentheorie (vgl. Rausch, 2010). Während die imaginären Objekte in jedem diskreten System zu finden sind, wird die Domäne des Modells durch die realen und Hilfsobjekte bestimmt. Reale Objekte sind zum einen die Fließobjekte, welche das System durchlaufen (beispielsweise Werkstücke oder Aufträge) und zum anderen die in ihren Ausprägungen sehr vielfältigen Ressourcen (beispielsweise Arbeiter oder Maschinen). Es ist nicht zwingend notwendig, Hilfsobjekte zu verwenden. Sie vereinfachen aber die Modellierung und sind in ihrer jeweiligen Domäne gebräuchlich. Ein einfaches Beispiel ist der Modus, der den Prozessschritt in einer bestimmten Konfiguration darstellt. Kann man einen Prozessschritt mit verschiedenen Ressourcen zu verschiedenen Zeiten ausführen, können diese Varianten als Modi bezeichnet werden. Es ist möglich, Modi als Blöcke, die einem Prozess zugeordnet sind zu modellieren. Es ist aber auch möglich, auf die Verwendung von Modi zu verzichten, indem für jeden Modus ein Prozess modelliert wird. Diese können dann im Verhaltensmodell durch Oder-Verknüpfungen verbunden werden.

Eine Assoziation zwischen zwei Objekten kann auch als Relation oder Beziehung bezeichnet werden. Beziehungen zwischen den Objekten können einfache Beanspruchungen sein, wie die Reservierung eines realen Objektes durch einen Prozess. Doch können Beziehungen durchaus auch komplexer und mit Attributen oder Bedingungen belegt sein. Ein Beispiel hierfür wäre eine Nachbedingung, welche einen Folgeprozess verpflichtet, die gleiche Ressource zu binden.

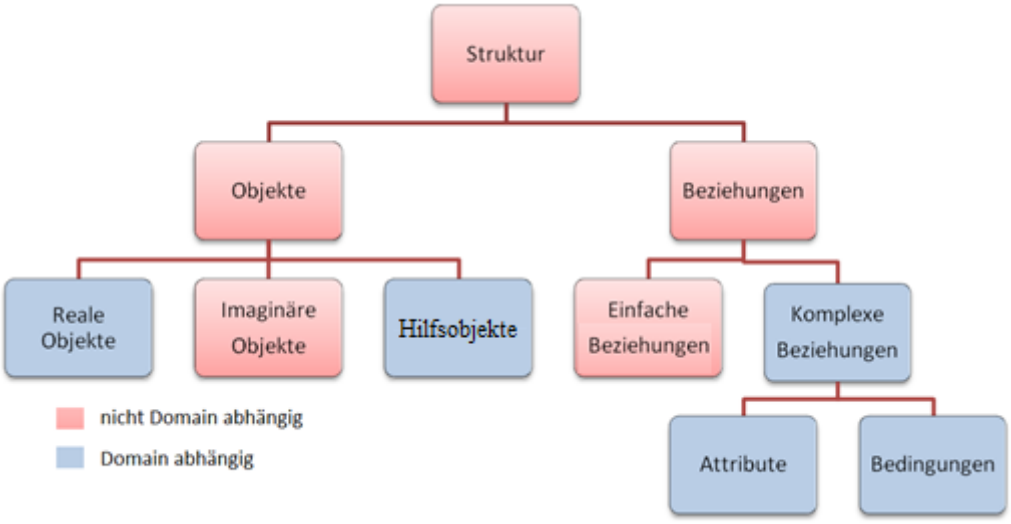


Abbildung 3-3: Übersicht Modellierung der Struktur

3.3.2 Ein Metamodell für die Struktur von Produktionsprozessen

In Kapitel 2.3 wurde gezeigt, wie mit SysML-Blockdefinitionsdiagrammen Szenarien von Metamodellen abgeleitet und modelliert werden können. In den folgenden Ausführungen wird ein grundlegendes strukturelles Metamodell für Produktionsszenarien besprochen. Abbildung 2-2 zeigt das strukturelle Metamodell für Produktionsprozesse. Während andere Modellbereiche, wie beispielsweise der Workflow, vom Informationsfluss bestimmt werden, bestimmt in der Produktion das *Entity* das Modellverhalten. Das *Entity* ist das zentrale Element eines Produktionsablaufes. Es stellt das Werkstück dar, das durch den Maschinenpark wandert und von dessen Elementen bearbeitet wird. Die Ereignisse innerhalb des Modells gehen bis auf Störungen vom *Entity* aus. Das *Entity* wird durch einen Ankunftsprozess in das Modell eingeschleust und verlässt es durch einen Abgangsprozess. Während es auf festgelegten Wegen wandert, veranlasst es Prozesse zu ihrer Ausführung. Dazu können sie Ressourcen benötigen. Zwischenzeitlich wird das *Entity* in Warteschlangen gelagert (Schönherr/Rose, 2010, S. 4).

Neben den domänenunspezifischen imaginären Objekten (Ankunftsprozess, Abgangsprozess, Prozess und Warteschlange) ist das *Entity* in der Produktion der Auftrag oder ein bestimmtes Produkt. Die domänenspezifischen Ressourcen unterteilen sich in Maschinen, Arbeiter, Transporter und sonstige Ressourcen. Wenn es möglich ist, einen Prozessschritt mit verschiedenen Ressourcen in unterschiedlichen Konfigurationen auszuführen, werden die Konfigurationen als das Hilfsobjekt Modus bezeichnet (Majohr, 2008, S. 26 f.). Für die Definition der Attribute werden im Metamodell Datentypen benötigt, die im SysML-Standard nicht definiert sind. Es ist möglich, diese im Metamodell als Enumeration zu definieren. Wichtige Enumerationen sind Zeiteinheiten, Einheiten für Geschwindigkeit und Einheiten für Verteilungen. Relation zwischen zwei Objekten gibt es in dem gezeigten strukturellen Metamodell zwischen Prozessen und Ressourcen, Modi und Ressourcen sowie zwischen Prozessen und Modi. Eine Relation kann durch den Stereotypen `ExtAsocciation` um Attribute und Bedingungen erweitert

werden. Die im Metamodell zugewiesenen Attribute sind eine Auswahl häufig verwendeter Eigenschaften. Je nach den Modellierungsspezialisierungen der jeweiligen Aufgabe kann eine Erweiterung des Metamodells um zusätzliche Attribute erfolgen. Diese Erweiterungen sollte der Entwickler bei der Modelltransformation berücksichtigen und umsetzen.

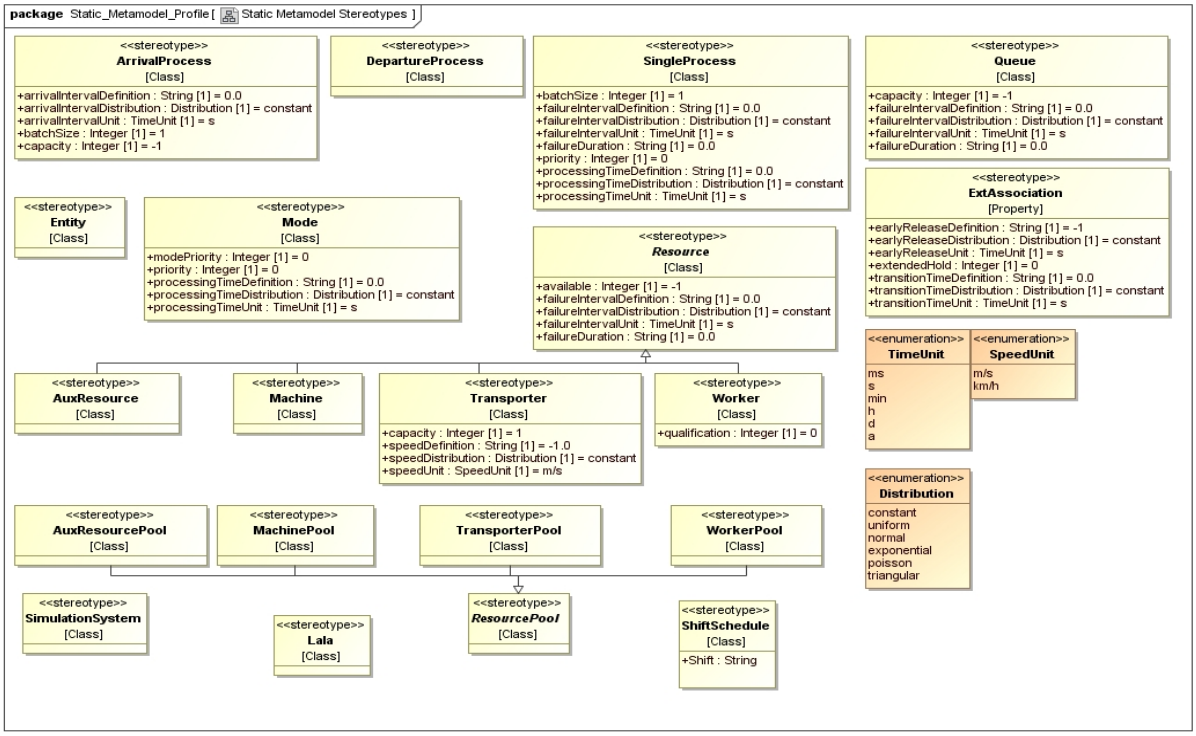


Abbildung 2-2: Metamodell für die Domäne der Produktion

Durch den **Ankunftsprozess (`ArrivalProcess`),** der auch oft als Quelle bezeichnet wird, treten *Entities* in das System ein. Daher steht der Ankunftsprozess immer mit einem *Entity* in Verbindung. Bei ein *Entity* und einem Ankunftsprozess wird die Relation – wie bei allen Verbindungen des *Entitys* – im Verhaltensdiagramm dargestellt (vgl. Kapitel 2.4). Der Ankunftsprozess hat einen internen Speicher, eine Batchgöße und eine Ankunftsrate, mit der die *Entities* das System betreten (vgl. Tabelle 3-1). Die Attribute des **Entitys** werden erst dann benötigt, wenn das algorithmische Modell genutzt wird und daher in diesem hergeleitet (vgl. Kapitel 3.6).

Tabelle 3-1: Definition des Ankunftsprozess (`ArrivalProcess`)

Attribut	Type	Syntax	Semantik
arrivalInterval Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF (Verteilungsfunktion) die bestimmt, in welchem Abstand *Entities* in das System eintreten.
arrivalInterval Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, in welchem Abstand *Entities* in das System eintreten
arrivalInterval Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF, die bestimmt, in welchem Abstand *Entities* in das System eintreten.
batchSize	Integer	0 .. 9999	Erst wenn die Größe batchSize erreicht ist, können die *Entities* den Prozess und nur gemeinsam verlassen.
capacity	Integer	0 ... 9999	Gibt an wie viele *Entities* im Prozess gespeichert werden können. Ist die Anzahl erreicht, können keine neuen Entities das System durch diesen Prozess betreten.

Durch den **Abgangsprozess (`DepartureProcess`)** verlassen die *Entities* das System. Das heißt, sie werden aus dem Modell entfernt. Der Abgangsprozess benötigt keine Attribute, da er einzig als Senke des Systems dient. In **Warteschlangen (`Queue`)** werden *Entities* gelagert, während sie den Maschinenpark durchlaufen. Warteschlangen haben eine Kapazität und können Ausfällen unterliegen (vgl. Tabelle 3-2).

Tabelle 3-2: Definition der Warteschlange (`Queue`)

Attribut	Type	Syntax	Semantik
failureInterval Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmen, in welchem Abstand ein Fehler bei der Warteschlange auftritt.
failureInterval Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, in welchem Abstand ein Fehler bei der Warteschlange auftritt.
failureInterval Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF, die bestimmt, in welchem Abstand ein Fehler bei der Warteschlange auftritt.
failureDuration	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, wie lange ein Fehler beim Auftreten anhält. Die Wahl der VF und Zeiteinheit werden aus failureIntervalDistribution und failureIntervalUnit übernommen.
capacity	Integer	0 ... 9999	Gibt an, wie viele Entitys in der Warteschlange gespeichert werden können. Ist die Anzahl erreicht, können keine neuen Entities die Warteschlange betreten.

Jede Aktion, die an einem *Entity* ausgeführt wird, wird als **Prozess (`SingleProcess`)** bezeichnet. Davon ausgenommen sind: das Speichern in einer Warteschlange, das Betreten des Systems durch den Ankunftsprozess und das Verlassen des Systems durch den Abgangsprozess. Zum Ausführen eines Prozesses können Ressourcen oder Ressource-Pools benötigt werden. Prozesse sind zeitliche Beanspruchungen des *Entity* oder von Ressourcen und können eine Zustandsänderung des *Entity*s oder der Ressourcen zur Folge haben. Auf diese Art gelangt das *Entity* in eine andere Bearbeitungsstufe bzw. eine Ressource in einen anderen Rüstzustand. Werden Prozesse im statischen Modell mit Ressourcen verbunden, bedeutet das, dass der Prozess während seiner Ausführung die verbundenen Ressourcen reserviert und sie zum Starten benötigt. Für die Beziehung ist es möglich eine Multiplizität anzugeben, die ausdrückt, wie viele Ressourcen der Prozess zum Starten benötigt. Weiterhin kann die Beziehung durch die erweiterte Assoziation mit zusätzlichen Attributen und Bedingungen belegt werden. Es ist

möglich die Beziehung als *Partproperty* oder Assoziation zu modellieren. Der Prozess hat eine Ausführungszeit, eine Batchgröße, eine Priorität und kann durch das Auftreten von Fehlern ausfallen (vgl. Tabelle 3-3).

Tabelle 3-3: Definition des Prozesses (`SingleProcess`)

Attribut	Type	Syntax	Semantik
processingTime Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, wie lange der Prozess für seine Ausführung braucht.
processingTime Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, wie lange der Prozess für seine Ausführung braucht.
processingTime Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF, die bestimmt wie lange der Prozess für seine Ausführung braucht.
batchSize	Integer	0 .. 9999	Erst wenn die Größe batchSize erreicht ist, können die Entities den Prozess und nur gemeinsam verlassen.
failureInterval Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, in welchem Abstand ein Fehler bei dem Prozess auftritt.
failureInterval Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, in welchem Abstand ein Fehler bei dem Prozess auftritt.
failureInterval Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF, die bestimmt, in welchem Abstand ein Fehler bei dem Prozess auftritt.
failureDuration	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, wie lange ein Fehler beim Auftreten anhält. Die Wahl der VF und Zeiteinheit werden aus failureIntervalDistribution und failureIntervalUnit übernommen.
priority	Integer	0 ... 9999	Gibt bei prioritätsorientierten Entscheidungen an, mit welcher Priorität der Prozess zu behandeln ist. Umso höher der Wert ist, desto höher ist die Priorität des Prozesses.

Ein in der Produktion typisches Element ist der **`Mode`**. Wie beschrieben können mit den Modi Prozesse in verschiedenen Konfigurationen ausgeführt werden. Beispielsweise ist es möglich einen Prozess mit zwei Arbeitern in fünf Minuten und mit einem Arbeiter in vierzehn Minuten durchzuführen. Die beiden Konfigurationen können nun als Modi modelliert werden. Ein Prozess kann beliebig viele Modi besitzen. Jeder Modus ist genau einem Prozess zugeordnet. Die Beziehung zwischen Modi und Prozess kann als *Partproperty* oder Assoziation modelliert werden. Wie beim Element-Prozess beschrieben, ist es möglich, den Modi in gleicher Weise Ressourcen zuzuordnen. Jeder Mode hat eine Priorität, eine Mode-Priorität und eine Bearbeitungszeit (vgl. Tabelle 3-4). Die Priorität bestimmt die Rangfolge bei der Konkurrenz um Ressourcen. Die Mode Priorität bestimmt die Rangfolge der verschiedenen Modi eines Prozesses.

Tabelle 3-4: Definition des Modes (`Mode`)

Attribut	Type	Syntax	Semantik
processingTime Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, wie lange der Mode für seine Ausführung braucht.
processingTime Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, wie lange der Mode für seine Ausführung braucht.
processingTime Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF, die bestimmt, wie lange der Mode für seine Ausführung braucht.
priority	Integer	0 ... 9999	Gibt bei prioritätsorientierten Entscheidungen an, mit welcher Priorität der Prozess zu behandeln ist. Je höher der Wert ist, desto höher ist die Priorität des Prozesses.
modePriority	Integer	0 ... 9999	Gibt bei prioritätsorientierten Entscheidungen an, wie hoch die Priorität des Modes gegenüber der eines anderen Modes ist. Je höher der Wert ist, desto höher ist die Priorität des Prozesses.

Ressourcen (`Resource`) werden vom Prozess beansprucht. Die Zuordnung ist beim Element-Prozess beschrieben. Ressourcen gibt es in verschiedenen Ausprägungen (mit verschiedenen Attributen) je nach Domäne. Die Ausprägungen der Ressource erben alle von der Klasse Ressource. Das heißt, dass sie Ressourcen sind und die Attribute der Ressource übernehmen. Alle Ressourcen stehen in einer bestimmten Menge zur Verfügung und können durch Fehler ausfallen (vgl. Tabelle 3-5).

Tabelle 3-5: Definition der Ressource (`Resource`)

Attribut	Type	Syntax	Semantik
failureInterval Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, in welchem Abstand ein Fehler bei der Ressource auftritt.
failureInterval Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, in welchem Abstand ein Fehler bei der Ressource auftritt.
failureInterval Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF, die bestimmt, in welchem Abstand ein Fehler bei der Ressource auftritt.
failureDuration	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, wie lange ein Fehler beim Auftreten anhält. Die Wahl der VF und Zeiteinheit werden aus failureIntervalDistribution und failureIntervalUnit übernommen.
available	Integer	0 ... 9999	Gibt an, wie viele Ressourcen des entsprechenden Typs bereitstehen.

Ressourcenpools (`ResourcePool`) können modelliert werden, um Ansammlungen von Ressourcen zu verdeutlichen. Es ist möglich Ressourcen über Assoziationen oder *Partpropertys* durch Ressourcenpools zu gruppieren. Die Zuordnung zu Prozessen erfolgt wie die Zuordnung von Ressourcen zu Prozessen. Ressourcenpools haben keine separaten Attribute.

Die Ressource **`Worker`** repräsentiert den Arbeiter, der eine bestimmte Qualifikation besitzen kann (vgl. Tabelle 3-6).

Tabelle 3-6: Definition der Ressource Arbeiter (*Worker*)

Attribut	Type	Syntax	Semantik
qualification	Integer	0 ... 9999	Teilt den Arbeiter durch einen Wert eine Qualifikation zu.

Die Ressource **Transporter** hat eine Kapazität, die angibt, wie viele *Entities* sie beladen kann, und eine Geschwindigkeit (vgl. Tabelle 3-7). Da das Modellierungskonzept die räumliche Modellierung nicht berücksichtigt, hat die Geschwindigkeit momentan keinen Einfluss, kann aber in an diese Ausarbeitung anschließenden Arbeiten berücksichtigt werden.

Tabelle 3-7: Definition der Ressource Transporter (Transporter)

Attribut	Type	Syntax	Semantik
speedDefinition	Distribution	Vgl. Tabelle 3.9	Werte für die VF (Verteilungsfunktion), die bestimmt, wie schnell sich der Transporter fortbewegt.
Speed Distribution	Distribution	Enumeration	Auswahl der VF, die bestimmt, wie schnell sich der Transporter fortbewegt.
speedInterval Unit	SpeedUnit	Enumeration	Auswahl der Geschwindigkeitseinheit für die VF, die bestimmt, wie schnell sich der Transporter fortbewegt.
capacity	Integer	0 .. 9999	Gibt an, wie viele Entities der Transporter transportieren kann.

Das Stereotyp **Machine** steht für die Ressource Maschine. Obwohl keine Attribute für Maschinen identifiziert werden konnten, wird dieses häufig verwendete Element zum besseren Modellverständnis durch den Anwender aufgenommen. **AuxRessource** repräsentiert die Verwendung einer beliebigen nicht klassifizierten Ressource. Die Klasse **SimulationSystem** erlaubt es, das System in verschiedene Subsysteme zu unterteilen. Diese können als *Partpropertys* hierarchisch gekapselt werden.

Das Element **ExtAssociation** ermöglicht es, Assoziationen zu erweitern. Im gezeigten Metamodell erweitert die ExtAssociation die Beziehung zwischen einem Prozess bzw. einem Modus und einer Ressource. Bei Herstellungsprozessen besteht die Möglichkeit, dass Ressourcen gehalten werden müssen, bis es möglich ist, den nächsten Prozess zu starten. Dieser Fall tritt ein, wenn zwischen zwei Prozessen keine Puffer vorhanden sind. Das Halten von Ressourcen ist nur zulässig, wenn keine vorzeitige Ressourcenfreigabe möglich ist und wird mit dem Attribut extendedHold umgesetzt. Ressourcen, die nicht über die komplette Prozesszeit bei der Verarbeitung benötigt werden, können durch das earlyRelease vorzeitig freigegeben werden. Zudem ist es möglich, zwischen Prozessen bzw. Modi und Ressourcen durch transitionTime Rüstzeiten zu definieren (vgl. Tabelle 3-8).

Tabelle 3-8: Definition der erweiterten Assoziation (`ExtAssociation`)

Attribut	Type	Syntax	Semantik
earlyRelease Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF, die bestimmt, nach welcher Zeit eine assoziierte Ressource freigegeben wird, obwohl der Prozess noch läuft . Für Assoziation zwischen einen Prozess und einer Ressource.
earlyRelease Distribution	Distribution	Enumeration	Auswahl der VF für das earlyRelease.
earlyRelease Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF vom earlyRelease.
extendedHold	Integer	0 .. 9999	Die mit extendedHold assoziierte Ressource muss die nächsten X Folgeprozessen des Entitys bedienen.
transitionTime Definition	Distribution	Vgl. Tabelle 3.9	Werte für die VF die bestimmt, wie lange die Rüstzeit bei der Assoziation des Prozess und der Ressource ist.
transitionTime Distribution	Distribution	Enumeration	Auswahl der VF für die transitionTime.
transitionTime Unit	TimeUnit	Enumeration	Auswahl der Zeiteinheit für die VF der transitionTime.

Neben den Stereotypen werden im Metamodell für Produktionsprozesse auch Enumerationen für Zeiteinheiten, Einheiten der Geschwindigkeit und Einheiten der Verteilungen festgelegt. Für Simulationsanwendungen werden Verteilungsfunktionen benötigt, deren Definition in der Enumeration **Verteilungen (`Distribution`)** erfolgt (vgl. Tabelle 3-9). Wird in einem Stereotyp eine Verteilung genutzt, hat der Anwender ein Attribut zum Wählen der hier definierten Verteilungsfunktion `distribution`, ein Feld zum Definieren der Werte der Verteilungsfunktion `definition` und ein letztes Feld zum Wählen der Einheit `unit`. In der folgenden Tabelle wird die jeweilige Syntax zum Definieren der Verteilungsfunktion festgelegt.

Tabelle 3-9: Definition der Enumeration Verteilung (`Distribution`)

Attribut	Type	Syntax	Semantik
Constant Distribution	String	constantValue	Berechnen eines Wertes mit Hilfe der Gleichverteilung.
Uniform Distribution	String	minValue_maxValue	Berechnen eines Wertes mit Hilfe der Uniformalverteilung.
Normal Distribution	String	meanValue_standard Deviation	Berechnen eines Wertes mit Hilfe der Normalverteilung.
Exponential Distribution	String	meanValue	Berechnen eines Wertes mit Hilfe der Exponentialverteilung.
Poisson Distribution	String	expectedValue	Berechnen eines Wertes mit Hilfe der Poissonverteilung.
Triangular Distribution	String	minValue_maxValue_ modeValue	Berechnen eines Wertes mit Hilfe der Triangularverteilung.

In der Enumeration **Zeiteinheit (`TimeUnit`)** werden Zeiteinheiten für die Modellierung der Simulationsanwendung zur Verfügung gestellt. Die genutzten Einheiten reichen von Millisekunde bis Woche (vgl. Tabelle 3-10).

Tabelle 3-10: Definition der Enumeration Zeiteinheiten (`TimeUnit`)

Attribut	Type	Syntax	Semantik
ms	Integer	0 ... 9999	Millisekunden
s	Integer	0 ... 9999	Sekunden
min	Integer	0 ... 9999	Minuten
h	Integer	0 ... 9999	Stunden
d	Integer	0 ... 9999	Tage
a	Integer	0 ... 9999	Wochen

Die Enumeration **Geschwindigkeit (`SpeedUnit`)** stellt die Maßeinheiten der Geschwindigkeit Meter pro Sekunde und Kilometer pro Stunde zur Verfügung (vgl. Tabelle 3-11).

Tabelle 3-11: Definition der Enumeration Geschwindigkeitseinheiten (`SpeedUnit`)

Attribut	Type	Syntax	Semantik
m/s	Integer	0 ... 9999	Meter pro Sekunde
Km/h	Integer	0 ... 9999	Kilometer pro Stunde

3.4 Das Verhaltensmodell

Im Verhaltensmodell wird das dynamische Verhalten der Objekte selbst und jenes zwischen ihnen beschrieben. Ein Beispiel aus der Produktion ist das Durchlaufen des Werkstückes durch den Maschinenpark. Die Modellierung des Verhaltensmodells kann nach der angestrebten Granularitätsstufe und abhängig davon, ob die Reihenfolge der Schritte eine Rolle spielen, klassifiziert werden (vgl. Abbildung 3-4). Für die Abbildung der Vorgänge, bei denen die Reihenfolge der Schritte eine Rolle spielt, wird das SysML-Aktivitätsdiagramm favorisiert, bei der Abbildung von Ressourcen mit dynamischem Verhalten das Zustandsdiagramm (Schönherr/Rose, 2010, S. 4).

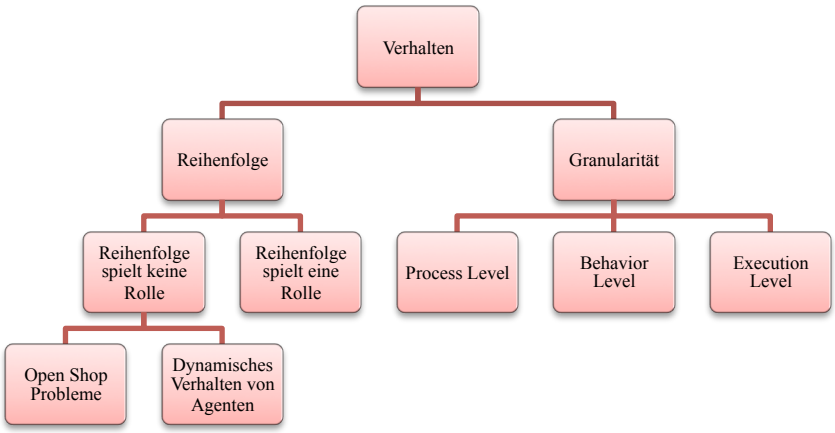


Abbildung 3-4: Übersicht Modellierung des Verhaltens (Schönherr /Rose, 2010, S. 4)

3.4.1 Unterteilung des Verhaltensmodells in Granularitätsstufen

Die Modellierung des Verhaltens kann in verschiedenen Granularitätsstufen durchgeführt werden. So beschreiben beispielsweise Pieper und Röttgers (2006, S. 46 ff.), wie sie die Abbildung des Verhaltens von *Workflows* in Granularitätsstufen unterteilen. Auch Störrle weist darauf hin, dass es möglich ist, Aktivitätsdiagramme in verschiedenen Granularitätsstufen zu beschreiben (Störrle, 2005, S. 194). Im Folgenden sollen Modelle der Produktion in verschiedene Granularitätsstufen unterteilt werden. Die Abbildung aller Stufen erfolgt mit SysML-Aktivitätsdiagrammen, deren Anwendung in Kapitel 2.4 erklärt wurde.

Die oberste Ebene soll als **Prozessebene** bezeichnet werden. Sie enthält eine Abbildung für die Abfolge der einzelnen Prozesse, die für den Durchlauf eines *Entities* durch den Maschinenpark steht. Jedes *Entity* des Szenarios hat einen Ablaufplan, der jeweils in einem Aktivitätsdiagramm auf der Prozessebene abgebildet wird. Das Beispiel in Abbildung 3-5 zeigt die Bearbeitung eines *Entities,* welches das System im Ankunftsprozess betritt, auf einer Maschine bearbeitet wird, anschließend abkühlt und das System wieder verlässt. Eigenschaften der Prozesse – wie Arbeitszeit oder benötigte Ressourcen – können den einzelnen Blöcken aus dem strukturellen Modell entnommen werden.

Auf dieser Ebene zu modellieren, ist für den Anwender fast immer ausreichend und wird von den meisten Simulatoren ausschließlich unterstützt. Es gibt jedoch auch Simulationswerkzeuge, die eine genauere Spezifikation benötigen. Beispielsweise beim Umsetzen von Simulatoren mit *Frameworks* wie JAMESII, muss der Entwickler feingranularer spezifizieren. Über das umgekehrte Gabelsymbol `CallBehaviorAction` können die Prozesse genauer definiert werden. Zum einen ist es so möglich, Prozesse auf derselben Granularitätsstufe zu kapseln (durch Zuordnung von untergeordneten Prozessstrecken), zum anderen ist es möglich, die nächste Granularitätsstufe zu modellieren.

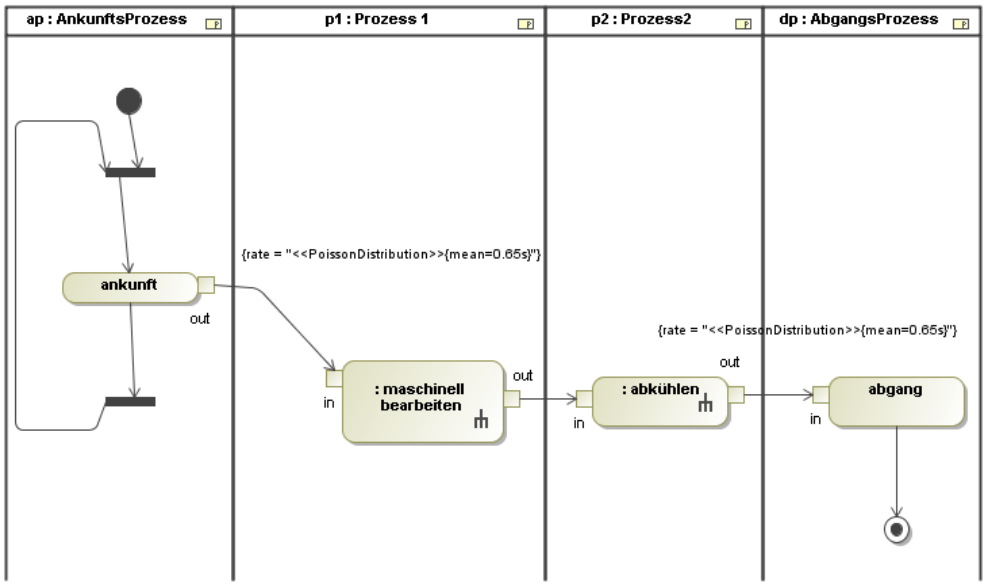


Abbildung 3-5: Beispiel Verhaltensmodell Granularitätsstufe I Prozessebene (Schönherr/Rose, 2010, S. 4)

Jedes Element der Prozessebene kann als ein Produktionsprozess klassifiziert werden (Ankunftsprozess, Abgangsprozess, Prozess, Warteschlange, Ressourcepool) (vgl. Kapitel 3.3). Diesem Prozess können nun in der nächsten Granularitätsstufe festgelegte Folgen von Verhaltensmustern zugeordnet werden. Die Stufe soll nachfolgend als **Verhaltensebene** bezeichnet werden. In dem gewählten Beispiel wird beschrieben, wie der Prozess `maschinell bearbeiten` vorgeht. In einem ersten Schritt bindet er die benötigten Ressourcen. Anschließend ist es möglich, das *Entity* gemäß der in den geschweiften Klammern angegebenen Zeit zu bearbeiten. Abschließend erfolgt die Freigabe der gebundenen Ressourcen (vgl. Abbildung 3-6).

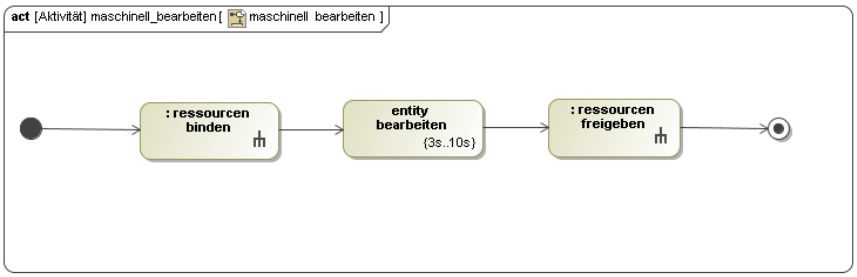


Abbildung 3-6: Beispiel Verhaltensmodell Granularitätsstufe II Verhaltensebene (Schönherr/Rose, 2010, S.4)

Während die ersten beiden Granularitätsebenen des Verhaltens deterministisch und ausführbar beschrieben werden, fehlt noch die genaue Beschreibung der Aktionen, wenn sie zur Ausführung kommen. Hierfür stellt die OMG ca. 40 Aktionen als Elemente der UML 2.0 bereit, die eine Grundlage für die Beschreibung von nicht-transientem Verhalten darstellen (Pieper/Röttgers, 2006, S. 47; vgl. Spezifikation UML). Die Aktivitäten werden in verschiedenen Büchern ausführlich besprochen (Weilkiens/Oestereich, 2006, S 161 ff.; vgl. Weilkiens, 2006 b; Rumbaugh et. al., 2004).

Die Beschreibung dieses fein-granularen Verhaltens erfolgt in der Stufe **Ausführungsebene**. Das angeführte Beispiel spezifiziert das Verhalten der Aktion `Ressourcen binden`. Zuerst wird die Anzahl der benötigten Arbeiter mit den zur Verfügung stehenden Arbeitern verglichen. Sobald genug Arbeiter zur Verfügung stehen, werden die benötigten Arbeiter reserviert und vom Ressourcenpool abgezogen (vgl. Abbildung 4). Während die Prozessebene vom Objektfluss des *Entitys* und die Verhaltensebene vom Kontrollfluss bestimmt werden, fließen in der Ausführungsebene Informationen über den Kontrollfluss. Mit dieser feingranularen Modellierung lässt sich das Verhalten eines Systems sehr detailliert beschreiben.

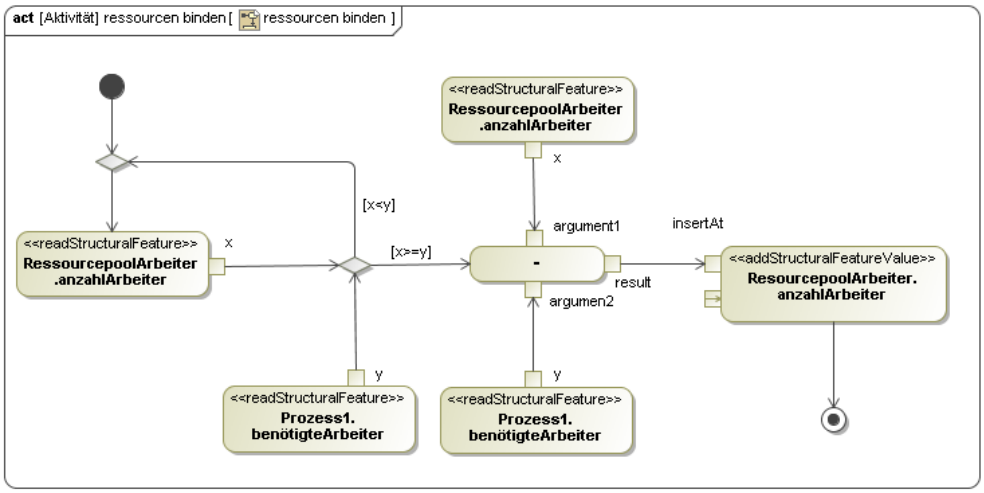


Abbildung 3-7: Beispiel Verhaltensmodell Granularitätsstufe III Ausführungsebene (Schönherr/Rose, 2010, S. 5)

Mit den drei festgelegten Granularitätsstufen lassen sich Verhaltensweisen der Produktion hierarchisch kapseln. So können sich wiederholende Verhaltens- oder Ausführungsmuster festgelegt und wiederverwendet werden. Während die Prozessebene individuell nach den Ablaufplänen eines bestimmten Szenarios modelliert wird, wiederholen sich in den unteren beiden Granularitätsstufen typische Muster. Auf diese Weise ist es möglich, jeden Prozess der Prozessebene als Produktionsprozess zu klassifizieren (Ankunftsprozess, Abgangsprozess, Prozess, Warteschlange, Ressourcenpool). Diesem kön-

nen nun in der Verhaltensebene festgelegte Verhaltensmuster zugeordnet werden. Den Elementen der Verhaltensebene können wiederum festgelegte Muster in der Ausführungsebene zugeordnet werden. Jede der drei Ebenen kann sich zusätzlich in eigene Zwischenstufen unterteilen. So ist es möglich, den Ablaufplan einer großen Fertigung in der Prozessebene mehrfach zu unterteilen, da sich bestimmte Muster, wie das Durchlaufen eines Bereiches des Maschinenparks, oft wiederholen.

3.4.2 Modellierungsmöglichkeiten innerhalb der einzelnen Granularitätsstufen

Um das Verhaltensmodell auf der ersten Granularitätsstufe darzustellen, nutzt diese Arbeit einen Auszug des SysML-Metamodells für Aktivitätsdiagramme. Alle in Abbildung 3-8 gezeigten Elemente können verwendet werden. Zusätzlich wurde im Metamodell dieser Arbeit das Stereotyp `Edge` des SysML-Metamodells um die stereotypisierten Eigenschaften `delay` und `probability` erweitert. Nach einem `DecisionNode` kann nun an den Kanten eine Wahrscheinlichkeit, die zu ihrer Nutzung führt, angegeben werden. Die `DelayedEdge` ist eine Alternative, die Prozesszeit im statischen Modell im Prozess anzugeben. Die genaue Definition der einzelnen Elemente kann neben der Erklärung im Kapitel 2.4.2 auch dem SysML-Metamodell entnommen werden (vgl. OMG, 2012, S. 91 ff.).

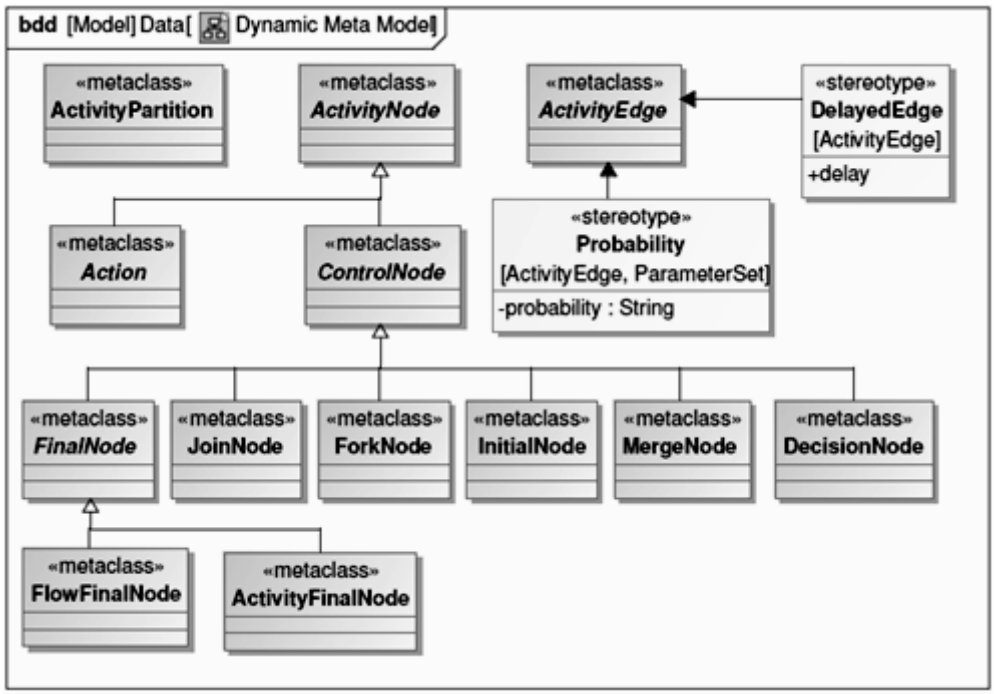


Abbildung 3-8: Metamodell der Verhaltensebene (Partzsch, 2010, S. 10)

Die Verhaltensphasen der Elemente können verschiedene Ausprägungen (Verhaltensmuster) haben (vgl. Tabelle 3-12). Diese Verhaltensmuster werden bei fast allen Simulatoren vom Entwickler festgelegt. Nach den Leitlinien dieser Arbeit müssen diese jedoch für den Anwender einsehbar sein. Daher werden sie bei Transformationen zu Simulatoren mit festgelegten Verhaltensmustern vom Entwickler

mit SysML-Verhaltensdiagrammen offengelegt. Dies ist natürlich nur möglich, wenn die Software dem Entwickler des Transformators den nötigen Einblick gewährt. Bei der Erstellung einer eigenen Software, die auf dem Konzept dieser Arbeit aufbaut, kann die Dokumentation der Verhaltensmuster mit SysML-Aktivitätsdiagrammen wie beschrieben umgesetzt werden.

Tabelle 3-2 zeigt einen Vorschlag für Verhaltensmuster der klassifizierten Elemente des statischen Modells. Diese üblichen Verhaltensmuster konnten den durchgeführten Untersuchungen von Simulatoren entnommen werden (vgl. Schönherr/Rose, 2009; vgl. Rehm, 2009). Es ist möglich, die Muster zu unterscheiden, je nachdem, ob es sich um eine ziehende (*pull*) oder schiebende (*push*) Steuerung handelt. Es soll noch einmal ausdrücklich darauf hingewiesen werden, dass jeder Simulator andere Verhaltensmuster nutzen kann. Innerhalb eines Simulationslaufes sind die Zuordnungen jedoch gleich.

Tabelle 3-12: Verhaltensmuster der Verhaltensebene (Schönherr/Rose, 2009, S. 5 f.; Rehm, 2009, S. 84)

	Verhaltensphase	Verhaltensmuster	
		aktiv	passiv
Quelle	Entity erzeugen	Erzeuge Entity	
	Entity weiterleiten	Entity weiterleiten	Entity freigeben
		Entity bedingt weiterleiten	Entity bedingt freigeben
Senke	Entity entgegennehmen	Hole Entity	Warte auf Entity
		Hole Batch	Warte auf Batch
	Entity vernichten	Vernichte Entity	
Puffer	Entity entgegennehmen	Hole Entity	Warte auf Entity
		Hole Batch	Warte auf Batch
	Entity einordnen	Entity einordnen	
	Entity weiterleiten	Entity weiterleiten	Entity freigeben
		Entity bedingt weiterleiten	Entity bedingt freigeben
Prozess	Entity entgegennehmen	Hole Entity	Warte auf Entity
		Hole Batch	Warte auf Batch
	Ressourcen binden	Ressourcen-Zustand abfragen	
		Ressourcen binden	Ressourcenzuweisung
	Aktion ausführen	abfragen des Entity-Zustands	
		bearbeiten des Entity-Zustands	
		Entity spalten	
		Entities synchronisieren	
	Entity weiterleiten	Entity weiterleiten	Entity freigeben
		Entity bedingt weiterleiten	Entity bedingt freigeben
Ressource	Rüsten	Rüsten	

Da auf der Ausführungsebene noch keine praktischen Arbeiten erfolgten, bleibt der vollständige Umfang der benötigten Aktionen offen.

3.5 Das Kontrollmodell

Im Kontrollmodell wird die Modellierung der Steuerung beschrieben, die alle zu treffenden Entscheidungen im System (Scheduling, Dispatching und Routing) berechnet. Im algorithmischen Modell werden die Algorithmen für das Scheduling, Dispatching und Routing und deren Nutzungsmöglichkeiten für den Anwender beschrieben. Die algorithmischen Entscheidungen können den vollständigen Systemzustand einbeziehen. Alle Elemente des Systems sind mit einem Monitor verbunden, der den kompletten Systemzustand bereitstellt. Wenn eine Entscheidung innerhalb der Simulation bei einem Element fällig ist, erhält der Kontroller die Informationen vom Monitor, führt einen Algorithmus aus und gibt das Ergebnis zum betreffenden Element zurück. Der Nutzer kann alle möglichen Algorithmen im algorithmischen Modell definieren, die der Kontroller dann ausführt. Im Kontrollmodell werden die Zusammenhänge zwischen dem Kontroller, dem Monitor, dem algorithmischen Modell und dem Basissystem beschrieben. Zudem werden die Aufgaben und die Architektur des Kontrollers und des Monitors festgelegt. Für den Anwender sind diese Informationen einzig informativ, da sie vom Entwickler definiert werden. Die Forschung hat bis jetzt keine Ansätze für die Konzipierung eines Kontrollmodells für Produktionssysteme hervorgebracht. Deshalb werden nun die Annahmen aus einer Studie von Simulationswerkzeugen, praktischen Erfahrungen und anhand abgeleiteter Konzepte aus der Literatur verwendet.

3.5.1 Vorbetrachtungen

Um die Eigenheiten der Steuerung in der Simulation zu verstehen, wurde in einer Diplomarbeit eine Auswahl an Simulationswerkzeugen der Produktion und Logistik hinsichtlich der relevanten Aspekte untersucht (vgl. Rehm, 2009). In der Arbeit wurden sieben Simulationswerkzeuge aus dem Bereich der Produktion betrachtet. Die Auswahl der Werkzeuge erfolgte durch Zuhilfenahme der Simulation Software Survey 2007 aus der Zeitschrift „*OR/MS Today*“ und der Marktübersicht von Simulationswerkzeugen aus Produktion und Logistik von Lindemann und Schmidt (2007). Es wurden nur bausteinorientierte, diskrete Simulatoren, die den Bereich der Produktion einschließen, gewählt. Folgende Simulationssysteme beinhaltet diese Untersuchung:

- Simul8, Trial Version 2009
- Process Simulator, Lite Version 2009
- AnyLogic, University Version 6.4.1
- Simcron Modeller, Version 3.1.4
- Flexsim, Evaluation Version 4
- DOSIMIS-3, Demo Version 5.0
- Plant Simulation, Student License Version 9.0.1

Simul8 ist ein kommerzielles Werkzeug, das seinen Schwerpunkt auf die Betrachtung von Geschäftsprozessmodellierung und insbesondere die Modellierung von Produktions- und Logistikszenarien legt. Steuerungsrelevante Aspekte erfolgen über eine Parametrisierung von speziellen Ereignispunkten der angebotenen Modellbausteine. Komplexere Algorithmen können in Simul8 mit einem *C-Derivat* geschrieben werden.

Process Simulator wird als Microsoft Visio Plug-In angeboten. Es eignet sich für Geschäftsprozessmodellierung, aber unterstützt auch spezielle Domänen mit *Manufacturing*, *Service* oder *Healthcare* Paketen. Process Simulator legt den Fokus kaum auf Steuerunsaspekte, weshalb es einen stark eingeschränkten Funktionsumfang aufweist, den der Nutzer nicht erweitern kann. So können nur einfache klassische *Dispatch*-Regeln und nur im Puffer-Element festgelegt werden.

AnyLogic versucht, den Umfang der Simulation sehr breit abzudecken. Es ist möglich, sowohl diskrete und kontinuierliche Systeme als auch Agentensysteme abzubilden. Zudem enthält es verschiedene Bibliotheken für unterschiedlichste Domänen. Die offene Architektur von Anylogic bietet an allen Bausteinen Ereignispunkte, die als Schnittstellen für vor- oder selbstdefinierte Java-Funktionen dienen.

Der **SimcronModeller** ist ein ereignisorientierter Simulator, der speziell für die Produktionsplanung konzipiert wurde. Es ist möglich, Steuerungsaspekte mit vorgefertigten Befehlsstrukturen der Objekte oder mittels separater Skripte zu modellieren. Als vorgefertigte Steuerungsmethoden stehen beispielsweise diverse Prioritätsregeln bereit. Benutzerdefinierte Verfahren können durch die Skriptsprache Tcl umgesetzt werden.

Flexsim ist ein Simulationswerkzeug für eine große Vielfalt diskreter Prozesse zahlreicher Industriezweige. Zudem bietet Flexsim auch Bibliotheken zum Bearbeiten kontinuierlicher Probleme, wie das Modellieren von Flüssigkeiten. Steuerungsmöglichkeiten werden bei Flexsim in allen Elementen an einer großen Menge von elementspezifischen Ereignispunkten angeboten. Während andere untersuchte Simulationssysteme (beispielsweise Simul8, Plant Simulation oder Anylogic) die Ereignispunkte für die Objekte zum Großteil vereinheitlichen (Ein- und Abgang eines *Entitie*s), sind die Ereignispunkte in Flexsim sehr elementspezifisch. Der Anwender kann an diesen Punkten eine Vielfalt vorgefertigter Regeln oder mit Flexscript selbstdefinierte Regeln nutzen.

DOSIMIS-3 ist ein zeitdiskretes Simulationswerkzeug, dessen Ausrichtung auf Produktionsprozessen liegt. Auch DOSIMIS-3 bietet Systembausteine, in denen Steuerungsfunktionen integriert sind. Zudem bietet es die Elemente `Arbeitsbereich`, das eine zentrale Verwaltung von Ressourcen ermöglicht, und `Transportsteuerung`, über das Routenentscheidungen von *Entities* zentral festgelegt werden können. Diese beiden Elemente haben den Charakter eines zentralen Kontrollers. Während andere Werkzeuge es möglich machen, über Funktionen einen eigenen Kontroller zu implementieren (beispielsweise Flexsim oder Simcron), ist bei DOSIMIS-3 die Infrastruktur für einen Kontroller sehr

nutzergeeignet vorbereitet. In der DOSIMIS-3-Dokumentation wird behauptet, komplett ohne die Notwendigkeit des Programmierens auszukommen bietet stattdessen eine Erweiterung der Standardfunktionalitäten über Anwendertabellen. Ob dieses Konzept wirklich ausreichend Möglichkeiten bietet um das Programmieren zu ersetzen, bleibt fraglich.

Plant Simulation ist ein Simulationsprogramm für diskrete Produktions- und Logistikprozesse. Mit zahlreichen Bibliotheken versucht es, bestimmte Teilbereiche wie beispielsweise Elektronikproduktion zu unterstützen. Die vielfältig vorgefertigten Objekte bieten Ereignispunkte, an denen eine große Anzahl an Steuermechanismen bereitstehen, aber auch mit der Sprache SimTalk erweitert werden können. Weiterhin bietet Plant Simulation einen `Broker` für die Verteilung von Ressourcen, der für diesen Aufgabenbereich als Kontroller genutzt werden kann.

In allen betrachteten Simulationsprogrammen sind das Kontrollmodell und das algorithmische Modell so eng miteinander verknüpft, dass kaum Unterscheidungen zu erkennen sind. Daher können beide Teilmodelle zusammen als Steuerungssystem betrachtet werden. Auch das Basissystem, das aus dem Struktur und Verhaltensmodell besteht, kann in den betrachteten Simulatoren nur schwer differenziert betrachtet werden. Die Differenzierung wird erschwert, weil in den Bausteinen der Simulatoren die Steuerungsmechanismen eingebunden sind. Trotzdem ist die in Abbildung 3-9 gezeigte Struktur zu erkennen, in der das Steuerungssystem Daten vom Basissystem nimmt und ihm Anweisungen zurückgibt. Im Konzept dieser Arbeit sind die Modelle eindeutig voneinander abgegrenzt. Während der Anwender über das algorithmische Modell Verfahren definiert, setzt das Kontrollmodell diese Verfahren um. Es bekommt Daten vom strukturellen- und Verhaltensmodell, berechnet vom Anwender im algorithmischen Modell festgelegte Algorithmen und gibt die Ergebnisse wieder an das Struktur- und Verhaltensmodell zurück (vgl. Abbildung 3-9).

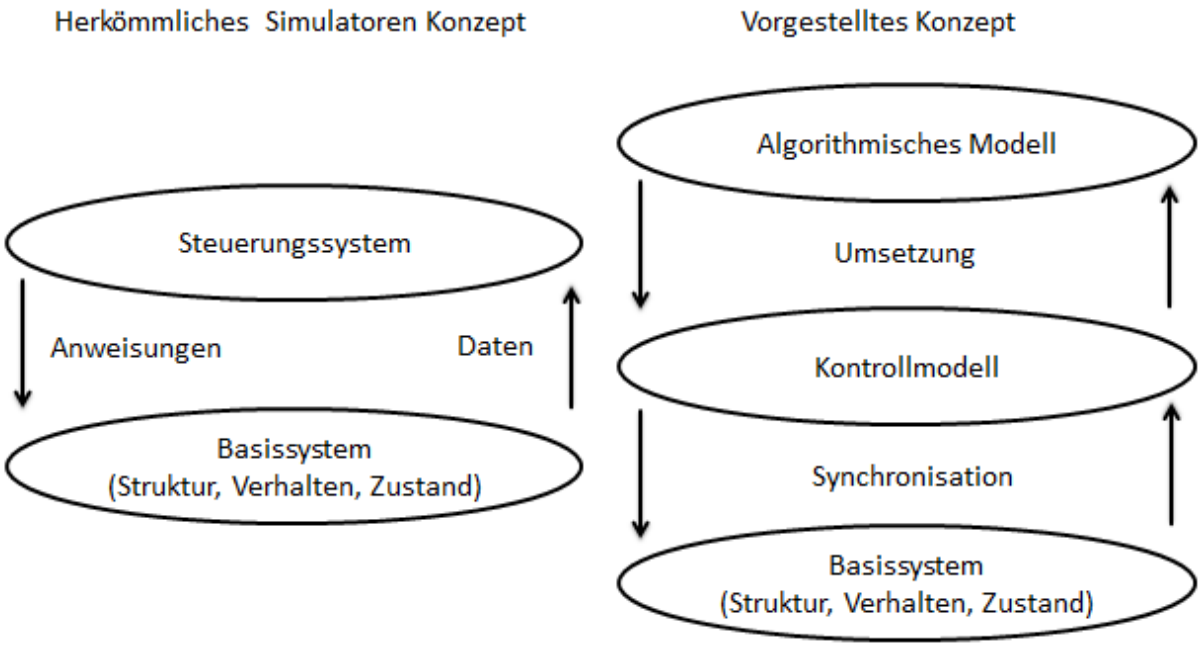


Abbildung 3-9: Kopplung des Steuerungssystems mit dem Basissystem

Die analysierten Simulatoren zeigen eine große Vielfalt an Steuerungsmechanismen. Diese lassen sich wie folgt klassifizieren. Die *Entities* werden zugeführt, zusammengefasst, sortiert und verteilt anhand von:

- Bedingung
- fester Zuweisung
- Zufall
- prozentualen Vorgaben
- Reihenfolge (nach Vorgabe oder reihum)
- Priorität (fix) der Nachfolgeelemente / aktuellen Objekte / *Entities*
- Algorithmen

Diese Muster kommen bei fast allen Modellbausteinen der Simulatoren vor.

Die Steuerung setzt, wie in Abbildung 3-10 gezeigt, auf dem Basissystem auf. Daher gibt es zu identifizierende (1) steuerungsrelevante Schnittstellen und Informationen (Kapitel 3.5.2), mit denen (2) Elemente des Kontrollmodells (Kapitel 3.5.3) mit Elementen des Basissystems kommunizieren. Anhand dieser Information werden dann (3) Berechnungen (Kapitel 3.5.4) durchgeführt und abschließend Informationen an das Basissystem zurückgegeben. Die folgenden Betrachtungen des Kontrollmodells sollen anhand dieser Dreiteilung vollzogen werden.

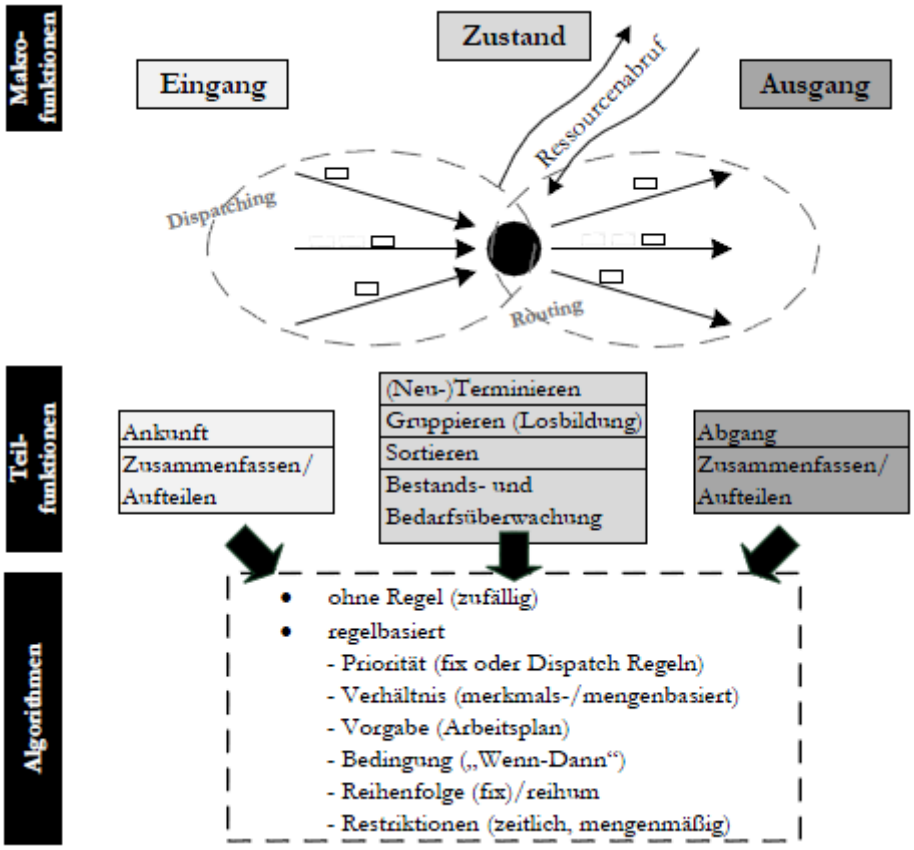


Abbildung 3-10: Schnittstellen bezogene Merkmale einer Steuerung (vgl. Rehm, 2009, S. 89)

3.5.2 Schnittstellen

Den Elementen des strukturellen Modells werden im Verhaltensmodell auf der zweiten Granularitätsebene Verhaltensphasen zugeordnet (vgl. Kapitel 3.4). Während diese Verhaltensphasen von einem Großteil der Simulatoren festgelegt werden, ist es möglich, diese bei Simulatoren-*Frameworks* (beispielsweise JAMES II (vgl. JAMES II, 2013)) oder einem selbst entwickelten Simulator vom Entwickler zu definieren. In den einzelnen Verhaltensphasen der Elemente können Schnittstellen vom Basismodell zum Steuerungsmodell zur Verfügung gestellt werden. In den folgenden Ausführungen wird ein einfaches Modell der Verhaltensebene gezeigt (vgl. Abbildung 3-11), das auf den Untersuchungen von Simulatoren zugrundeliegender Arbeiten aufbaut (vgl. Schönherr/Rose, 9009; vgl. Rehm, 2009). Es wäre auch möglich, dieses Modell mit vier SysML-Aktivitätsdiagrammen auf der Verhaltensebene zu beschreiben, was aus Platzgründen vereinfacht wurde. Es wird gezeigt, wie dem Beispielmodell Kommunikationsabläufe und somit Schnittstellen zwischen Basismodell und Steuerungsmodell zugeordnet werden können.

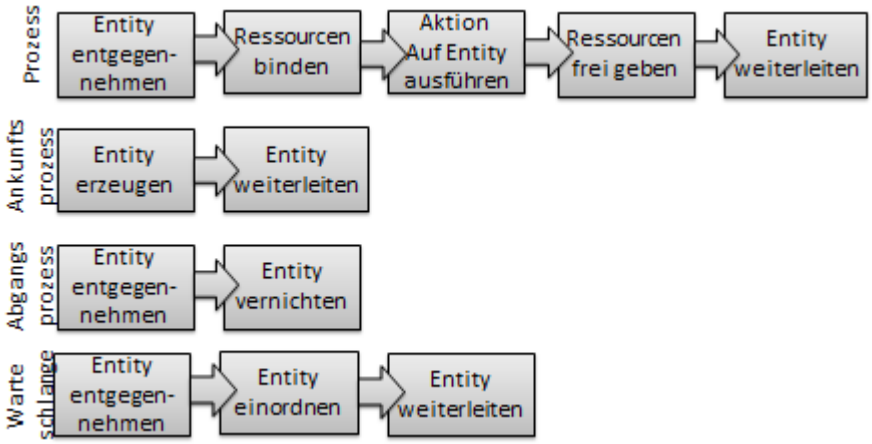


Abbildung 3-11: Beispiel für Verhaltensphasen

Die Verhaltensphasen werden nun nach möglichen Kommunikationsabläufen mit dem Kontrollmodell geordnet (vgl. Abbildung 3-12). Die Abläufe der Kommunikation können unterteilt werden in `Ressourcen Reihenfolgebildung`, `Ressourcen Freigabe`, `Zustandsänderung`, `Reihenfolgebildung und Freigabe`. Da die Kommunikationsabläufe algorithmische Bearbeitungen des Kontrollmodells veranlassen, beziehen sich die Namen für die Kommunikationsabläufe auf die Verfahrensklassifikation des algorithmischen Modells (vgl. Kapitel 3.6). Die genauen Kommunikationsabläufe und die dabei übertragenen Informationen werden im Kommunikationsmodell definiert (vgl. Kapitel 3.9).

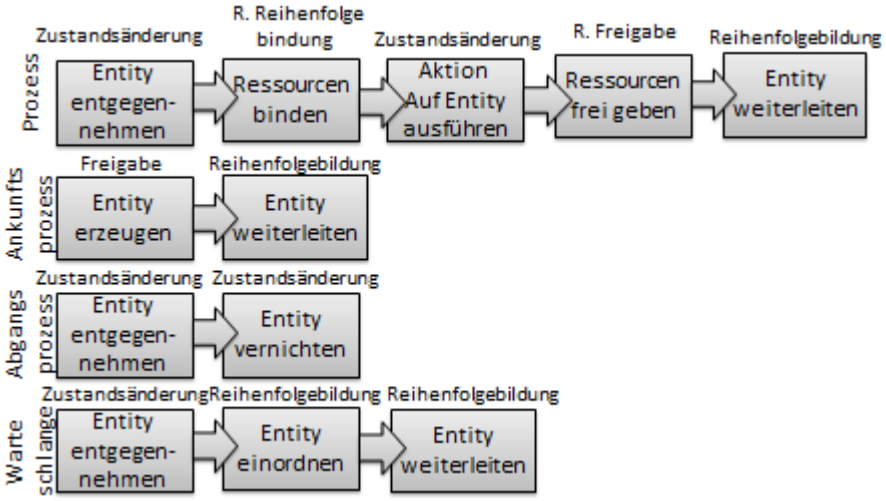


Abbildung 3-12: Abgeleitete Kommunikationsabläufe anhand von Verhaltensphasen

3.5.3 Der Monitor und der Kontroller

Das Kontrollmodell enthält die beiden Elemente Monitor und Kontroller. Der Monitor sammelt Informationen über das Modell und erstellt damit parallel zur Simulation ein Datenmodell. Das Datenmodell spiegelt den Zustand des Systems wider. Der Zweck des Monitors besteht darin, dem Kontroller bei Bedarf alle Informationen bereitzustellen, die er zum Ausführen von Berechnungen benötigt. Das Datenmodell des Monitors hat somit den Umfang der vom Kontroller benötigten Informationen. Die Kommunikation des Monitors und Kontrollers kann in einem Sequenzdiagramm beschrieben werden (vgl. Abbildung 3-29). Immer wenn der Kontroller zum Ausführen einer Berechnung aufgefordert wird, fordert er die benötigten Informationen vom Monitor ein. Der Monitor erhält bei jedem Kommunikationsvorgang zwischen Basismodell und Steuerungsmodell nötige Zustandsinformationen. Alle Kommunikationsabläufe, die Zustandsänderungen durchführen, werden vom Monitor genutzt, um das Datenmodell konsistent zu halten. Alle anderen Abläufe stellen dem Monitor Zustandsinformationen zur Verfügung und fordern den Kontroller zum Durchführen von Berechnungen auf.

Der Kontroller bekommt eine Anfrage während Verhaltensphasen, in denen Entscheidungen getroffen werden müssen. Anhand der vom Anwender definierten Algorithmen im algorithmischen Modell fordert der Kontroller notwendige Informationen vom Monitor an, führt Berechnungen aus und gibt die Ergebnisse als Anweisungen zurück. Daher müssen die im algorithmischen Modell definierten Aktionen im Kontroller implementiert werden. Anhand der Aktionen ist es auch möglich, neue Algorithmen zusammenzusetzen. Neben algorithmischen Entscheidungen zählen auch *Deadlock*-Vermeidungsstrategien zu den Aufgaben des Kontrollers. *Deadlocks* werden in vielen Bereichen der Informatik wie Betriebssystemen, Datenbanken, Rechnernetzen oder der technischen Informatik behandelt. Die Literatur zur Vermeidung von *Deadlocks* ist zahlreich (vgl. Tannenbaum, 2009). Die Implementierung des Kontrollers und Monitors ist nicht Teil dieser Arbeit und Bestandteil von Folgearbeiten. Die Funktionsweise des Monitors sowie des Kontrollers lässt sich am besten mit Zustandsdiagrammen darstellen. Abbildung 3-13 zeigt exemplarisch ein einfaches Konzept eines Kontrollers.

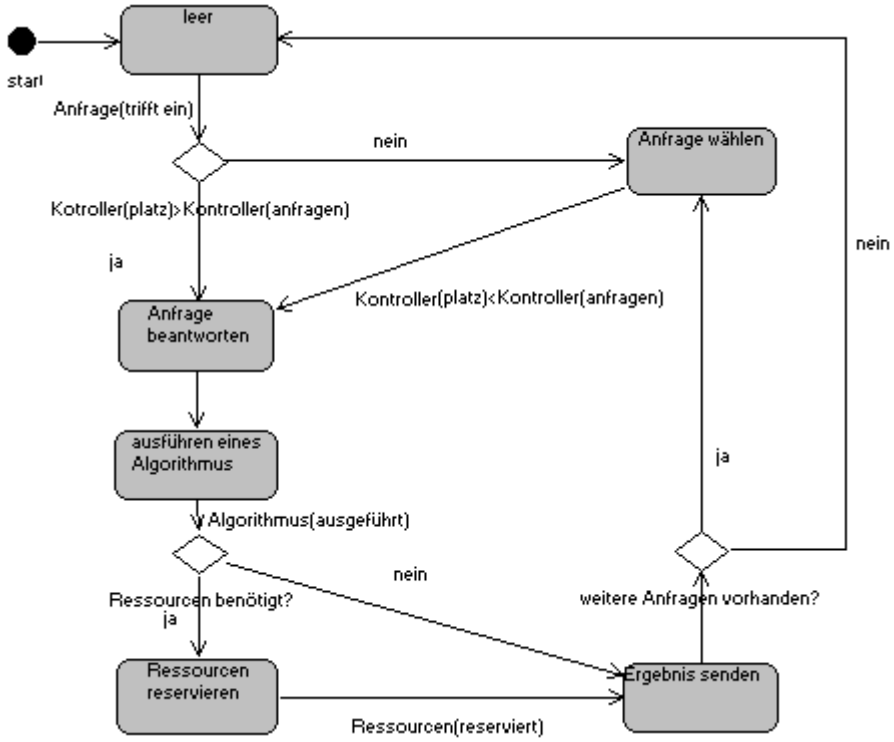


Abbildung 3-13: Zustandsmodell Kontroller

3.5.4 Umsetzungen des Kontrollmodells bei Modelltransformationen

Bei einer der beiden einleitend erwähnten Untersuchungen wurden sieben Simulationsprogramme hinsichtlich ihres Modellierungskonzeptes auf besonderen Bezug der Steuerkomponenten betrachtet. Alle Systeme haben entweder vorgefertigte Steuerregeln oder aber Schnittstellen für vom Anwender selbst zu implementierende Funktionen (vgl. Rehm, 2009). Die vom jeweiligen Programm angebotenen Funktionen können einen solchen Umfang haben, dass der Anwender einen vollständigen Kontroller nachimplementieren muss (vgl. Pappert, 2007). Die angebotene Breite der steuerungsrelevanten Funktionalität ist bei den verschiedenen Simulationsprogrammen sehr unterschiedlich. Einige Programme wie beispielsweise Process Simulator oder AnyLogic bieten wenig vorgefertigte Steueralgorithmen an, stellen dem Anwender dafür aber die Möglichkeiten bereit, benutzerdefinierte Funktionen zu erstellen. Andere Programme wie Plant Simulation bieten dem Nutzer eine große Anzahl vorgefertigter Steuerimplementierung, die aber auch äußerst flexibel gestaltet werden können (vgl. Rehm, 2009, S. 77).

Bei einer Modelltransformation soll der Anwender das ihm im algorithmischen Modell bereitgestellte Spektrum zum Großteil nutzen können. Bei der Modellierung erstellt der Anwender ein Szenario in dem er die Struktur, das Verhalten und die zugehörigen Algorithmen gestaltet. Anschließend verbindet das Kontrollmodell das Basissystem mit dem algorithmischen Modell (vgl. Abbildung 3-9). Das vom

Anwender zur Modellierung verwendete Teilmodell, das aus Struktur-, Verhaltens- und algorithmischen Modell besteht, wurde in diesem Konzept möglichst flexibel gestaltet, um dem Anwender einen weitgehend freien Einsatz von Algorithmen zu gestatten. Die Flexibilität und Vielfalt der Algorithmen ist – wie gerade beschrieben – bei den verschiedenen Simulationsprogrammen sehr unterschiedlich. Um die Verbindung zwischen Basismodell und algorithmischem Modell herstellen zu können, muss das Kontrollmodell fehlende Funktionalitäten des jeweiligen Simulationsprogramms ausgleichen und überbrücken. Aus den Betrachtungen entstehen drei mögliche Fälle, die jeweils bei der Modelltransformation unterschiedlich umgesetzt werden müssen.

1. Das Simulationswerkzeug stellt die vom algorithmischen Modell benötigten Steuerfunktionalitäten innerhalb und zwischen den angebotenen Systemelementen bereit.

2. Das Simulationswerkzeug stellt die benötigten Steuerfunktionalitäten nicht innerhalb und zwischen den Systemelementen bereit, bietet aber die Möglichkeit, einen Kontroller innerhalb des Werkzeuges zu erstellen, der den benötigten Steuerfunktionalitäten genügen kann.

3. Das Simulationssystem bietet nicht die Möglichkeiten, um den benötigten Funktionsumfang der Steuerfunktionalitäten abzudecken.

In den folgenden Betrachtungen wird beschrieben, wie das Kontrollmodell die Infrastruktur des Simulationsprogrammes nutzen bzw. erweitern kann, um das Basismodell mit dem algorithmischen Modell zu verbinden.

Transformation des Kontrollmodells bei ausreichender Steuerfunktionalität des Simulators

Wenn der Simulator, auf dem das Modell abgebildet werden soll, die benötigten Steuerfunktionalitäten innerhalb und zwischen den angebotenen Systemelementen bereitstellt, kann das komplette Modell mittels Regeln zur Transformation übersetzt werden. Die Verbindungen des Steuermodells mit dem Basismodell werden in diesem Fall vom Simulator bereitgestellt. Die Zuordnungen der Algorithmen zum Basismodell können einzig durch Transformationsregeln übertragen werden. In diesem Fall kann die Transformation vollständig durchgeführt werden. Dies ist der komfortabelste Fall für den Entwickler der Modelltransformation, aber auch der unwahrscheinlichste. Während der Aufwand für die Entwicklung eines Kontrollers entfällt, muss für die Transformation die Funktionsweise der Verfahren und deren Struktur im Austauschformat genau evaluiert werden.

Transformation des Kontrollmodells bei Simulationswerkzeugen mit intern implementierbarem Kontroller

Weiterhin kann es sein, dass der Simulator die benötigte Steuerfunktionalitäten innerhalb und zwischen den angebotenen Systemelementen nicht bereitstellt, aber dafür die Möglichkeit einen internen Kontroller zu implementieren, wie es beispielsweise Flexsim ermöglicht. In diesen Fall muss der Entwickler die Infrastruktur des Simulators zwischen dem Kontroller und dem Basismodell genau ermitteln, um anschließend einen Kontroller mit den intern bereitgestellten Möglichkeiten zu entwickeln.

Der Aufwand besteht in der Entwicklung eines Kontrollers mit der vom Simulator genutzten Sprache, die unter Simulationsprogrammen in der Regel stark variieren und daher oft neu angeeignet werden muss (vgl. Schönherr, 2008; Wenzel, 2009, S. 3). Ein Risiko besteht darin, dass ein interner Kontroller nicht vollständig entwickelt werden kann, da der Simulator nicht genug Möglichkeiten bereithält. Der bis dahin entstandene Entwicklungsaufwand ist dann verschwendet. Wenn der Kontroller unter den genannten Bedingungen entwickelbar ist, kann das vollständige Modell durch Transformationsregeln übertragen werden, die Anwendung bleibt aber auf das spezielle Programm beschränkt.

Transformation des Kontrollmodells bei nicht ausreichender Steuerfunktionalität des Simulators

Eine weitere Möglichkeit besteht darin, einen externen Kontroller zu entwickeln. Um dies durchführen zu können, muss der Simulator eine Schnittstelle zum Austausch von Daten während der Simulation anbieten. Eine solche Schnittstelle war bei fünf der sieben betrachteten Programme in Rehms Studie vorhanden (vgl. Rehm, 2009). Beim Vorhaben der Entwicklung eines externen Kontrollers für ein Simulationsprogramm kann das Vorhandensein einer solchen Schnittstelle schnell evaluiert werden. Am Lehrstuhl Modellierung und Simulation der Bundeswehr Universität München wurde beispielsweise ein externer Kontroller für das Simulationsprogramm Factory Explorer entwickelt.

Abbildung 3-14 zeigt eine mögliche Struktur für einen externen Kontroller. Der Kontroller und der Simulator werden über eine Ereignisliste (*Event List*) synchronisiert. In der Liste werden *Events* des Simulators definiert, bei deren Auftreten der Kontroller aufgerufen wird. Tritt ein solches *Event* ein, stoppt der Simulator und ergänzt eine Austauschdatei (*User.dll*), über die der Simulator mit dem Kontroller-Interface kommuniziert. Der Kontroller erhält die nötigen Informationen aus der Austauschdatei und führt die notwendigen Berechnungen unter Zuhilfenahme verschiedener Komponenten durch. Abschließend gibt das Kontroller-Interface die berechneten Ergebnisse als Anweisungen über die Austauschdatei an den Simulator zurück, der dann bis zum nächsten Auftreten eines vereinbarten Ereignisses weiter läuft.

Der Vorteil der gewählten Architektur ist, dass sie für verschiedene Simulatoren wiederverwendbar ist. Bei der Transformation der Modelle müssen die Ereignispunkte des Simulators mit denen des Modells synchronisiert werden. Die Verfahren des algorithmischen Modells werden bei dieser Vorgehensweise mit dem Kontroller abgestimmt.

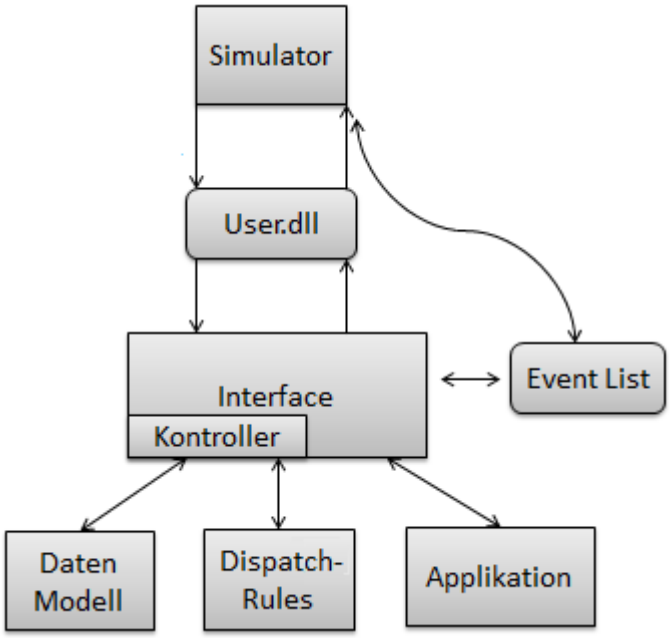


Abbildung 3-14: Architektur eines externen Kontrollers

3.6 Das algorithmische Modell

Um das algorithmische Modell verständlich zu präsentieren, wird es in drei Teilen vorgestellt. Der erste Teil erklärt, wie Algorithmen in das vorgestellte Konzept eingebunden und vom Anwender genutzt werden können. Im zweiten Teil des Abschnitts soll ein Spektrum an Algorithmen ermittelt werden, welches das Aufgabenfeld der Produktionssteuerung möglichst umfassend abdeckt. Um dies zu erreichen, muss ermittelt werden, welche Aufgabengebiete die Produktionssteuerung umfasst und wie sie klassifiziert werden können. Anschließend werden die klassifizierten Teile der Aufgabengebiete mit gebräuchlichen Algorithmen gefüllt. Im dritten Teil dieses Abschnitts werden Algorithmen in ihre Teilstücke/Elemente zerlegt. Die einzelnen Elemente werden später im Kontroller als Programmcode hinterlegt. So können durch Neukombination bestehender Teilstücke oder durch die Kombination von neuen und bestehenden Elementen neue Algorithmen mit reduziertem Aufwand entwickelt werden. Die Darstellung der zerlegten Verfahren mit SysML-Aktivitätsdiagrammen wird im ersten Teil des Abschnitts besprochen. Bei der Betrachtung der Elemente sind Ähnlichkeiten und Wiederverwendbarkeit besonders interessant. Diese Eigenschaften lassen sich besonders gut in abgegrenzten Teilgebieten betrachten. Daher wird das im zweiten Teil des Abschnittes hergeleitete Spektrum der Produktionssteuerung genutzt, um das betrachtete Feld möglichst gleichmäßig abzudecken. Die Arbeitsergebnisse dieses Kapitels sind an Ergebnisse einer zugrundeliegenden Masterarbeit angelehnt (vgl. Krög, 2011).

3.6.1 Umsetzung des algorithmischen Modells

Für die Umsetzung des algorithmischen Modells wird grundlegend zwischen der Perspektive des Anwenders und der des Entwicklers unterschieden.

Darstellung des algorithmischen Modells für den Anwender

Um das algorithmische Modell nutzen zu können, wird ein gesondertes Blockdefinitionsdiagramm erstellt, in dem alle benötigten Algorithmen abgebildet werden. Wird das angebotene Modellierungswerkzeug TOPCASED *Engineer* genutzt, steht hier ein besonderes Paket mit der Bezeichnung `Algorithm` bereit. Bei der Nutzung des zugeordneten BDD wird dem Anwender eine Auswahl an eingebundenen stereotypisierten Algorithmen bereitgestellt. Der Anwender kann nun die gewünschten Algorithmen auswählen, welche dann als stereotypisierte Blöcke in das BDD eingebunden werden. Abbildung 3-15 zeigt auf der linken Seite ein kleines Szenario mit vier Elementen. Auf der rechten Seite der Abbildung ist ein weiteres BDD, das das Algorithmische Modell, das aus dem Algorithmus `CONWIP1` besteht, abbildet. Der Block `CONWIP1` ist vom Stereotypen `CONWIP` abgeleitet und enthält zwei Attribute `rule`, `capacity` sowie zwei als Part Property zugeordnete Elemente `Warteschlange` und `Prozess1`. Jeder Algorithmus hat vom Anwender zu spezifizierende Attribute und wird über *Partproperties* den Elementen des Szenarios zugeordnet.

Das *Constant Work in Process* Verfahren (CONWIP) stellt einen konstanten Umlauflagerbestand sicher, indem neue *Entities* nur dann in den CONWIP-Abschnitt gelassen werden, wenn ein *Entity* mit ungefähr dem gleichen Arbeitsaufwand den CONWIP-Abschnitt verlässt. Der CONWIP-Abschnitt umfasst eine Teilstrecke des Systems, die durch den CONWIP reguliert wird. Diese bezieht sich nicht auf einen Prozess, sondern auf einen festgelegten Bereich mit zugeordneten Elementen. Die Reihenfolge mit der die *Entities* das System betreten, wird oft mit Prioritätsregeln ermittelt (Jodlbauer, 2008, S. 225 ff.). Der Nutzer kann bei CONWIP also über Attribute festlegen, mit welchem Prioritätsregelverfahren die Reihenfolge, mit der die *Entities* das System betreten, ermittelt wird (rule) und wie viele *Entities* den CONWIP-Abschnitt betreten dürfen (capacity). In dem gezeigten Beispiel wird die Reihenfolge der *Entities* mit `FIFO` bestimmt und es können nicht mehr als fünf *Entities* gleichzeitig im CONWIP-Abschnitt sein, der im Beispiel aus den Elementen `Warteschlange` und `Prozess 1` besteht. In TOPCASED *Engineer* können die Elemente vom Anwender einfach durch *drag-and-drop* dem jeweiligen Element als *Partproperty* zugeordnet werden.

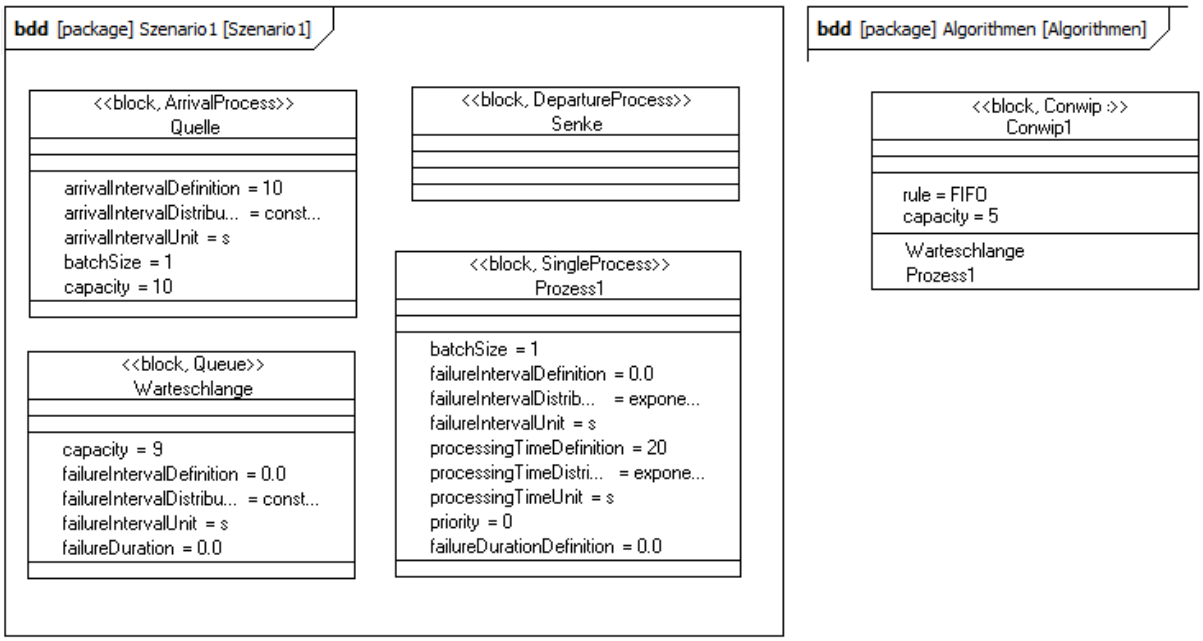


Abbildung 3-15: Beispiel der Zuordnung von Algorithmen zu Elementen

Darstellung des algorithmischen Modells für den Entwickler

Im dritten Teil dieses Abschnittes werden Algorithmen in ihre Teilstücke / Elemente zerlegt (vgl. Kapitel 3.6.3). Die einzelnen Elemente der Verfahren werden später im Kontroller als Programmcode hinterlegt. So können durch Neukombination bestehender Teilstücke oder durch die Kombination von neuen und bestehenden Elementen abgeänderte oder auch neue Algorithmen mit reduziertem Aufwand entwickelt werden. Hauptsächlich ist diese Funktionalität für den Entwickler gedacht, der Elemente nachträglich in SysML und im Kontroller hinzufügen kann, um mit ihnen neue Algorithmen zu entwickeln. Wenn das Konzept gereift ist und eine große Menge von Verfahren abgebildet wurde und somit viele Elemente bereitstehen, dann kann auch der Anwender das Konzept nutzen, um neue Algorithmen zu entwickeln.

Um die Algorithmen darzustellen, eignen sich SysML-Verhaltensdiagramme mit stark gekürztem Umfang. Die Komponenten eines Algorithmus werden durch `CallBehaviorActions` dargestellt und durch den SysML-Informationsfluss verbunden. Es werden der `InitialNode` und der `FinalNode` zum Starten und Beenden der Aktivitäten genutzt. Für die Darstellung von logischen Verzweigungen eignen sich `Decision` und `MergeNodes`. Abbildung 3-16 zeigt als Beispiel die Abbildung des im letzten Abschnitt beschriebenen Verfahrens `CONWIP`. Nachdem ein *Entity* das System betritt, wird der Algorithmus initalisiert (`get initial values`). Anschließend wird ein der CONWIP-Strecke vorliegender *Schedule* anhand einer vom Nutzer gewählten Prioritätsregel geordnet (`use priorityRule`). Nachdem der *Schedule* gespeichert wurde (`store schedule`), wird überprüft (`check CONWIP-capacity for entity`), ob sich mehr *Entities* in der CONWIP-Strecke befinden als vom Anwender über das Attribut `capacity` erlaubt ist. Ist die

CONWIP-Strecke noch nicht an ihre Kapazitätsgrenze gestoßen, kann das *Entity* diese betreten. Andernfalls wird der Algorithmus neu initialisiert. Das Beispiel zeigt, dass Algorithmen Elemente mit Attributen beinhalten, die vom Anwender später spezifiziert werden sollen. Das für den Anwender entstehende *Stereotyp* setzt sich also aus Elementen zusammen, die für den Anwender verborgen bleiben (`get initial values, store schedule`) oder als Attribut im Stereotyp angezeigt werden (`use priorityRule, check CONWIP-capacity for entity`).

Diese Methodik eignet sich auch für die Entwicklung sehr komplexer Verfahren. Abbildung 3-17 zeigt das Giffler-Thompson-Verfahren (vgl. Krög, 2011, S.33). Es bestimmt hier die jeweils frühesten Start- oder Endzeitpunkte der einzuplanenden *Entities*. Sind zwei *Entities* gleichzeitig einplanbar, kann dieser Konflikt mittels ergänzender Prioritätsregeln gelöst werden. In den Untersuchungen war es möglich, alle betrachteten Algorithmen mit diesen Sprachelementen und der vorgestellten Methodik abzubilden (vgl. Krög, 2011).

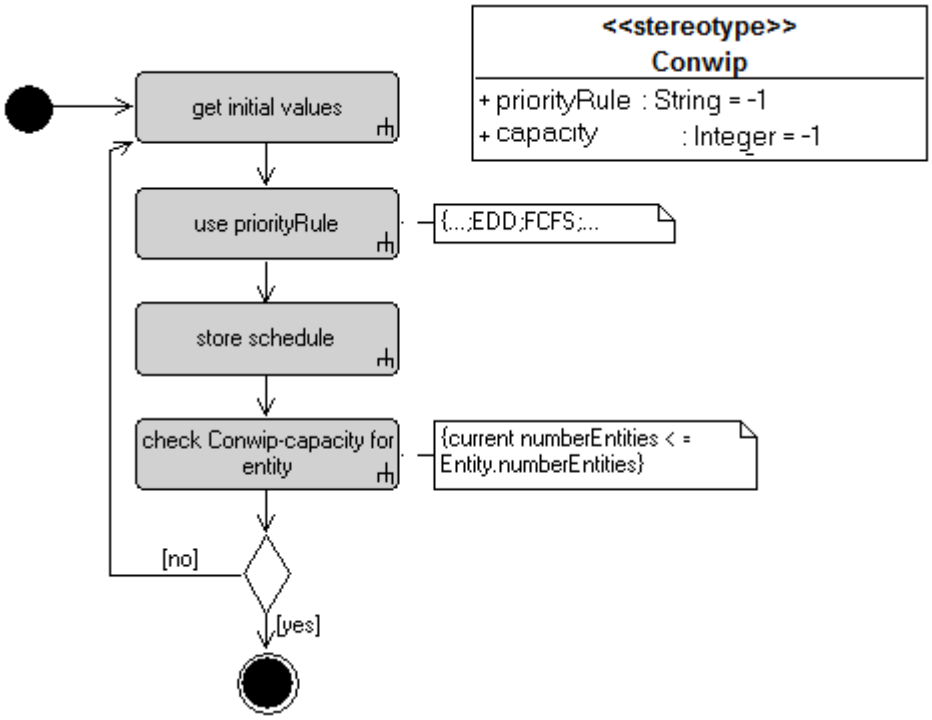


Abbildung 3-16: Darstellung der Zusammensetzung des CONWIP Verfahrens (vgl. Krög, 2011, S. 86)

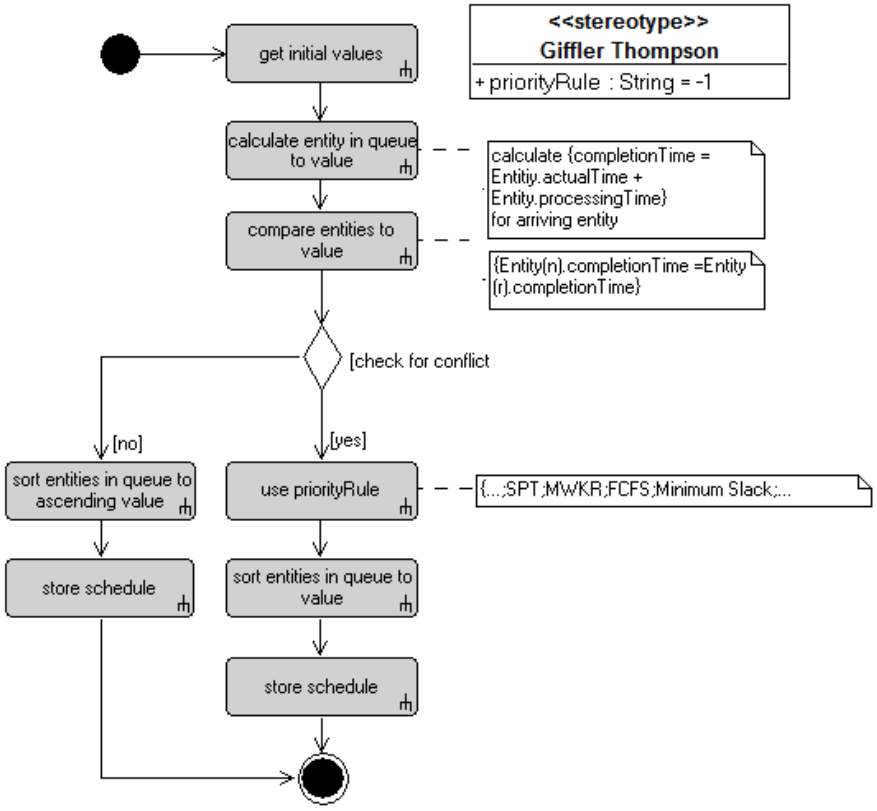


Abbildung 3-17: Darstellung der Zusammensetzung des Giffler-Thompson-Verfahrens (vgl. Krög, 2011, S. 33)

3.6.2 Einordnung und Abgrenzung von Aufgabenbereichen der Produktionssteuerung

Die Produktionssteuerung ist für die Erfüllung der Aufgaben zuständig, um die Ziele der PPS umzusetzen (Hügens, 2008, S. 419). Grundsätzlich existieren zwei Produktionssteuerungsstrategien: das *Pull*-Prinzip, bei dem die Aufträge durch die Fertigung „gezogen" werden und das *Push*-Prinzip bei dem die Aufträge vom jeweiligen Element zum nächsten „geschoben" werden. Beim *Pull*-Prinzip erfolgen die Entscheidungen oft in jedem Knoten dezentral (lokal). Bei dem *Push*-Prinzip regelt des Öfteren eine zentrale (globale) Steuerung die Entscheidungen. Lokale Entscheidungen können direkt beim Element (mit Hilfe einer kleinen Steuereinheit) getroffen werden, während globale Entscheidungen eine zentrale Steuereinheit benötigen. Oft erfolgt eine Vermengung von Systemen mit globalen und lokalen Entscheidungen (hybrides System) (Höch, 1998, S. 212). Diese Arbeit betrachtet lokale und globale Entscheidungen differenziert.

Weiterhin haben sich vier grundlegende Punkte durchgesetzt, nach denen Steuerungsüberlegungen in der Literatur unterschieden werden (Adam, 1998, S. 617).

- Die Zuordnung von Ressourcen zu Werkstätten / Steuereinheiten / Teilsystemen (Kapazitätssteuerung),
- die Planung von innerbetrieblichen Auftragsgrößen (Auftragserzeugung),
- die Festlegung der Reihenfolge von Aufträgen (Reihenfolgeplanung),
- die Freigabe von Aufträgen (Auftragsfreigabe).

Traditionelle Konzepte der Produktionsplanung und –steuerung

In der traditionellen Ansicht soll aufgrund der Auftragslage und der Mengen der zeitmäßige Produktionsablauf geplant werden. Dabei sind Ressourcen einzuplanen, freizugeben und zu überwachen. Bei Abweichungen ist es notwendig, Maßnahmen zu deren Regulierung zu ergreifen.

Kiener und auch Hackstein sehen sowohl die Reihenfolgeplanung als auch die operative Überwachung als Aufgaben der Produktionssteuerung. Kiener erläutert besonders die Reihenfolgenplanung, die auch als Feinterminierung der Maschinenbelegungsplanung oder Ablaufplanung bezeichnet wird. Deren Aufgabe sieht er in der Verteilung von Aufträgen auf Ressourcen, die unter Restriktionen abgearbeitet werden sollen (Kiener et al., 2009, S. 154 ff.; Hackstein, 2002 S. 3-28).

Das Konzept von Mathar und Scheuring ordnet die Auftragsveranlassung und Auftragsüberwachung der Produktionssteuerung zu. Die Auftragsveranlassung besteht hier unter anderem aus der Auftragsfreigabe und Arbeitsverteilung, die Auftragsüberwachung aus Kapazitätsüberwachung und Fortschrittsüberwachung (Mathar/Scheuring, 2009, S. 132).

Adam misst der Produktionssteuerung lediglich die Kapazitätssteuerung zu, während er die anderen Funktionen der Produktionsplanung zuordnet (Adam, 1998, S. 597).

Moderne Ansätze für die Produktionsplanung und –steuerung

Ein neues Modell ist das Aachener PPS-Modell. Hier werden in der Produktionsprogrammplanung die Aufträge der Kunden als Absatzplanung anhand ihres Ressourcenbedarfs auf Machbarkeit vorgeplant. In der Produktionsbedarfsplanung werden die Durchlaufterminierung, der Materialbedarf und die Kapazitätsbestimmungen geprüft. Die Fertigungs- und Bestellaufträge liegen nun nach Menge und Terminierung vor. Sie werden in der Fremdbezugsplanung sowie Eigenfertigungsplanung ausgeplant und gesteuert (Scherer, 1998).

Das Konzept der *Advanced Planning and Scheduling Systems* (APS) kann als modular aufgebautes Softwaresystem gesehen werden, das hierarchische Planung mit der Sichtweise einer global koordinierten Einheit vorgibt. Wie auch beim Aachener PPS-Modell sind die Aufgabenbereiche der Produktionssteuerung nicht eindeutig zuzuordnen (vgl. Kiener et al., 2009, S. 289; Yang, 2004, S. 70).

Computer Integrated Manufacturing (CIM) konzentriert sich neben der Produktionsprogrammplanung auf die Managementphilosophie zur Führung der organisatorischen und personellen Leitfähigkeit. Die Bereiche der Auftragserzeugung und Auftragsfreigabe werden hier klar aus dem Bereich der Produktionssteuerung ausgeschlossen (Scheer, 1990, S. 2 ff.; Abramovici/Schulte, 2004, S. 5 ff.).

Lödding beschreibt in seinem Ansatz sowohl die Interdependenzen der Aufgaben der Fertigungssteuerung als auch logistische Zielgrößen. Im Modell ist es möglich, durch jeden der vier Aufgabenbereiche Ist- und Plan-Größen als Stellgrößen festzulegen. Aus den Abweichungen der Stellgrößen ergeben sich Regelgrößen, die sich auf die logistischen Werte Termintreue, Auslastung, Bestand und Durchlaufzeit auswirken (Lödding, 1998, S. 195)

Tabelle 3-13: Zuordnung des Aufgabengebietes der Produktionssteuerung von verschiedenen Autoren (vgl. Krög, 2011, S. 16)

- trifft nicht zu	X trifft zu	O nicht exakt definierbar

	Aufgaben der Produktionssteuerung				
Autor	**Auftrags-erzeugung**	**Auftrags-freigabe**	**Reihenfolge-bildung**	**Kapazitäts-steuerung**	**Konzept**
Kiener et al.	-	-	x	x	PPS
Mathar / Scheuring	-	x	x	x	PPS
Hackstein	-	-	x	x	PPS
Adam	-	-	-	x	PPS
Scheer	o	o	o	o	Aachener PPS
Yang / Kiener et al.	o	o	o	o	APS
Scheer / Abramovici / Schulte	-	-	o	o	CIM
Lödding /Nyhuis	o	x	x	x	Aufgaben-orientierter Ansatz
Zusammenfassung	-	x	x	x	Zusammenfassung

Tabelle 3-13 zeigt eine Zusammenfassung der beschriebenen Konzepte der Produktionssteuerung und welche Ansätze welchem Aufgabengebiet der Produktionssteuerung zugeordnet werden. Der Zusammenfassung ist zu entnehmen, dass die Auftragserzeugung nicht der Produktionssteuerung zugeordnet wird, sondern eher der Planungsebene. Die Auftragsfreigabe wird nicht durchgehend der Produktionssteuerung zugeordnet, aber in den Betrachtungen dieser Arbeit trotzdem mit einbezogen. Gerade die Belastung einzelner Arbeitssysteme spielt für den abgegrenzten Untersuchungsgegenstand der Dissertation eine Rolle und wird von den Aufgabefreigabeverfahren berücksichtigt. Bei der Reihenfolgebildung zeigt sich deutlich, dass sie in die Produktionssteuerung eingeordnet wird. Daher ist sie auch Gegenstand der folgenden Betrachtungen. Auch die Kapazitätensteuerung wird von etlichen Autoren und Konzepten im Bereich der Produktionssteuerung eingeordnet. Doch werden bei der Kapazitätssteuerung hauptsächlich kurzfristige Abweichungsregelungen betrachtet, die der zeitnahen Steuerung zugeordnet werden. Lokale und globale Prioritätsregelverfahren sind für diese Arbeit die zeitnahesten Verfahren und zugleich auch Gegenstand der Verfahrensklassen der Reihenfolgebildung. Daher wird die Kapazitätssteuerung in dieser Arbeit nicht separat betrachtet. Die nach den Betrachtungen relevanten Reihenfolgebildungs- und Auftragsfreigabeverfahren sollen nun identifiziert und klassifiziert werden, um einen Überblick der für diese Arbeit relevanten Verfahren zu erstellen. Die Übersicht besteht aus globalen sowie lokalen Verfahren, um den anfänglichen Betrachtungen gerecht zu werden.

Die Auftragsfreigabe unterteilt sich in die zeitliche, ressourcenberechtigte und bestandsregelnde Auftragsfreigabe. Die zeitliche Auftragsfreigabe gibt die Aufträge sofort nach ihrem Eingang zur Bearbeitung frei. Doch können Bestand, Auslastung und Durchlaufzeit nicht direkt von dieser Methode der Auftragsfreigabe beeinflusst werden. Bei der ressourcenbedingten Auftragsfreigabe werden die Aufträge erst dann freigegeben, wenn alle benötigten Ressourcen vorhanden sind, was zu langen Wartezeiten führen kann. Die bestandsregelnde Auftragsfreigabe unterteilt sich in Verfahren mit zeitlichen und ressourcenbedingten Aspekten. Für diese Arbeit sind vor allem die Verfahren der bestandsregelnden Auftragsfreigabe relevant, da sie die Belastung der Arbeitssysteme berücksichtigen und auch oft in der Praxis eingesetzt werden (Lödding, 1998, S. 311). In Abbildung 3-18 sind wichtige Verfahren der bestandsregelnden Auftragsfreigabe aufgezeigt und nach ressourcenbedingten und zeitlichen Gesichtspunkten klassifiziert.

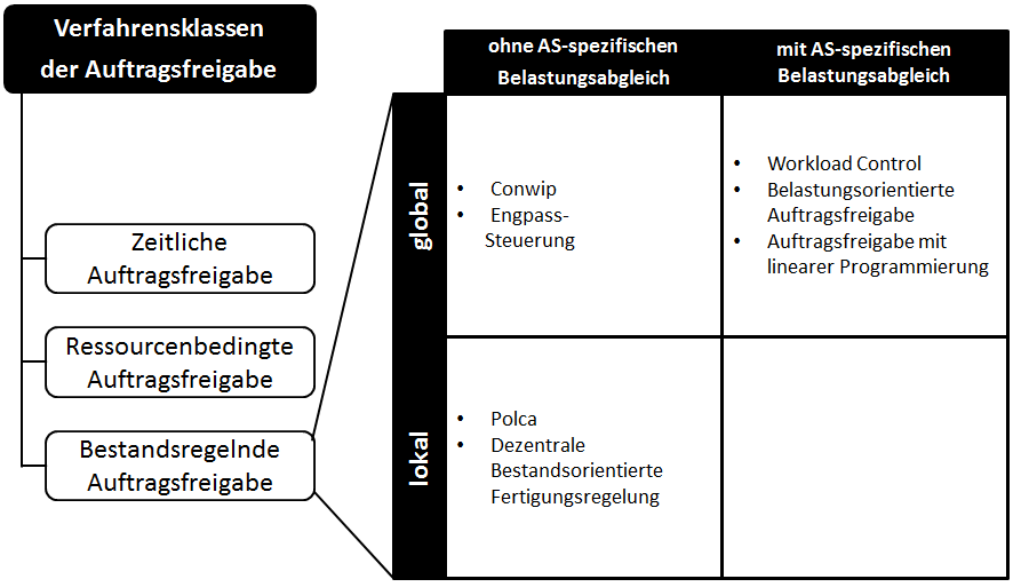


Abbildung 3-18: Klassifizierte Verfahren der Auftragsfreigabe (vgl. Lödding, 1998)

Verfahren der Reihenfolgebildung unterteilen sich grundsätzlich in exakte und heuristische Verfahren. Da die Verfahren der Reihenfolgebildung sehr mächtig, aber auch komplex sind, werden sie eher für die umfangreicheren globalen Probleme genutzt. Auch komplexe Verfahren der künstlichen Intelligenz wie neuronale Netze oder Agentensysteme eignen sich für die Lösung solcher Probleme. Prioritätsregelverfahren können für weniger schwierige lokale Probleme aber auch komplex und erweitert für globale Probleme genutzt werden (vgl. Abbildung 3-19).

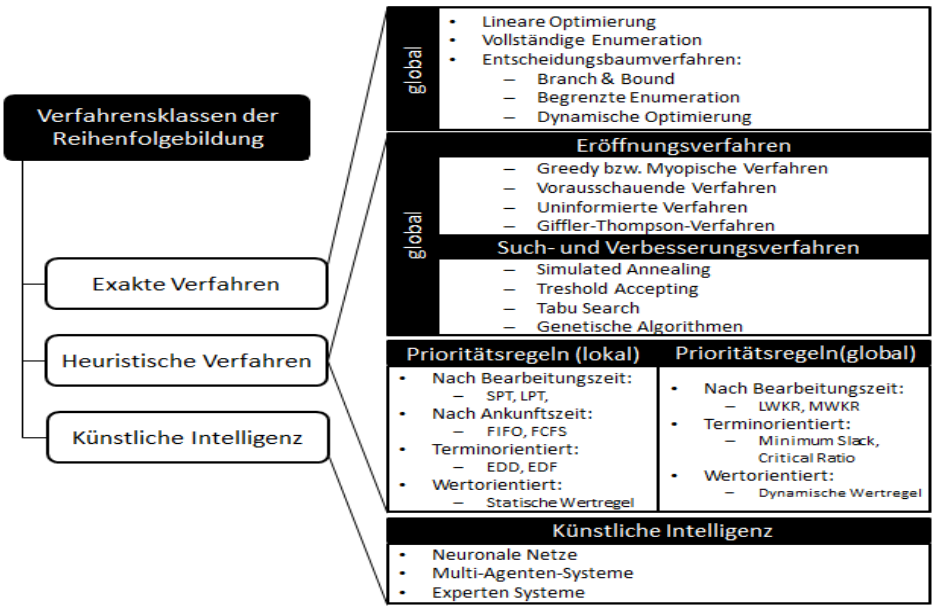


Abbildung 3-19: Klassifizierte Verfahren der Reihenfolgebildung (vgl. Krög, 2011, S.20)

3.6.3 Betrachtung verschiedener Algorithmischer Verfahren und ihrer Bestandteile

Im folgenden Teil sollen nun verschiedene Algorithmen genauer betrachtet und in ihre Bestandteile zerlegt werden, um die Elemente dann durch Neukombination zur Erstellung neuer Algorithmen verwenden zu können. Somit sind für die folgenden Betrachtungen Ähnlichkeiten und Wiederverwendbarkeit der Elemente besonders interessant. Um das Aufgabenfeld der Produktionssteuerung möglichst umfassend abzudecken, ist der folgende Abschnitt daher nach dem vorhergehend hergeleiteten Spektrum der Produktionssteuerung unterteilt. Die einzelnen Algorithmen werden in den folgenden Ausführungen nicht ausführlich, sondern nur sehr kurz im Sinngehalt erklärt. Zur genaueren Betrachtung sind für alle Verfahren Quellen, in denen die Algorithmen ausführlich besprochen werden, angegeben. Diese Arbeit beschäftigt sich lediglich mit der Identifizierung von Gemeinsamkeiten der Algorithmen, geordnet nach dem zuvor hergeleiteten Spektrum der Produktionssteuerung. Die hergeleiteten Gemeinsamkeiten der (Elemente) sind Grundlage für die erste Überprüfung der Konzeption des algorithmischen Modells und der prototypischen Entwicklung einer Kontrollerlogik. Die vorgestellten Ergebnisse beruhen auf einer dieser Arbeit zugrundeliegenden Masterarbeit (vgl. Krög, 2011).

Lokale Prioritätsregel-Verfahren der Reihenfolgebildung

Beim Verfahren der Reihenfolgebildung werden zuerst die weniger komplexen Prioritätsregelverfahren betrachtet. Bei diesen Verfahren wird jeweils das *Entity*, das einem betrachteten Attribut am besten genügt, weitergeleitet. In diesem Feld wurden die folgenden fünf Verfahren betrachtet: *First Come*

First Served (Weiterleitung nach Wartezeit), *Shortest Processing Time* (Weiterleitung anhand der kürzesten Prozesszeit), *Longest Processing Time* (Weiterleitung anhand der längsten Prozesszeit), *Earliest Due Date* (Weiterleitung anhand des geforderten Fertigungsdatums) und Prioritätsregelverfahren mit statischen Werteregeln (Weiterleitung anhand des höchsten Materialwertes).

Sämtliche Verfahren ähnelten sich in ihrer Zusammensetzung (vgl. Tabelle 3-14). Alle Verfahren der folgenden Betrachtungen starten mit einer Initialisierung (`get initial value`) und enden mit der Speicherung der erzeugten Reihenfolge (`store schedule`). Das *First Come First Served* (FCFS) Verfahren erhöht nach der Initialisierung den numerischen Wert (`numerical value`) aller *Entities* des betroffenen Elements um eins, um die Warteschlange anschließend aufsteigend (`sort entities in queue to ascending value`) anhand des Reihenfolge Attributes zu sortieren. Die anderen Algorithmen der Verfahrensklasse nutzen lediglich das aufsteigende (`sort entities in queue to ascending value`) oder absteigende (`sort entities in queue to descending value`) Sortieren anhand eines bestimmten Attributes.

Tabelle 3-14: Zusammensetzung Lokaler Prioritätsregel Verfahren der Reihenfolgebildung (vgl. Krög, 2011, S. 67)

- trifft nicht zu	X trifft zu	O nicht exakt definierbar

Aktionsschritt	SysML-Aktion	FCFS	SPT	LPT	EDD	Statische Wertregel
Initialisierung	get initial values	x	x	x	x	x
numerisch Wert erhöhen	increase numeral value of all entities	x	-	-	-	-
aufsteigend sortieren	sort entities in queue to ascending value	x	x	-	x	-
absteigend sortieren	sort entities in queue to descending value	-	-	x	-	x
Schedule speichern	store schedule	x	x	x	x	x

Globale Prioritätsregel Verfahren der Reihenfolgebildung

Die globalen Prioritätsregelverfahren sind etwas komplexer als die lokalen. Die Regel *Minimum Slack* ordnet die *Entities* nach der aktuell noch verbleibenden Zeit bis zum Liefertermin abzüglich der ausstehenden Bearbeitungszeiten (vgl. Bloech et al, 2003, S. 307). Bei der Regel *Critical Ratio* wird die verbleibende Zeit bis zum Liefertermin mit der ausstehenden Bearbeitungszeit ins Verhältnis gesetzt (vgl. Bloech et al, 2003, S 307). Die *Least Work Remaining* Regel priorisiert das *Entity* mit der niedrigsten noch verbleibenden Restbearbeitungszeit, die Regel *Most Work Remaining Time* das mit der höchsten (Waldner, 2008, S. 84). Bei der dynamischen Werteregel wird das *Entity* mit dem höchsten derzeitigen Produktwert priorisiert, dadurch wird der Fortschritt in der Produktionskette berücksichtigt (Wannenwetsch, 2009, S. 570).

Neben dem absteigenden und aufsteigenden Sortieren wird hier eine SysML-Aktion benötigt (`calculate value of all entities in queue`), die den aktuellen Wert eines *Entity*s anhand eines bestimmten Kriteriums (ausstehende Bearbeitungszeit) berechnet. Wie bei den lokalen Prioritätsregeln kann auch bei den globalen Prioritätsregeln erkannt werden, dass oft verschiedene Algorithmen die einzelnen SysML-Aktionen verwenden und sich lediglich auf andere Attribute beziehen.

Tabelle 3-15: Zusammensetzung Globaler Prioritätsregel Verfahren der Reihenfolgebildung (Krög, 2011, S. 68)

- trifft nicht zu	X trifft zu	O nicht exakt definierbar

Aktionsschritt	SysML-Aktion	Minimum Slack	Critical Ratio	LWKR	MWKR	Dyn. Wertregel
Initialisierung	get initial values	x	x	x	x	x
numerisch Wert erhöhen	calculate value of all entities in queue	x	x	x	x	-
aufsteigend sortieren	sort entities in queue to ascending value	x	x	x	-	-
absteigend sortieren	sort entities in queue to descending value	-	-	-	x	x
Schedule speichern	store schedule	x	x	x	x	x

Komplexe Verfahren der Reihenfolgebildung

Außer den Prioritätsregelverfahren gibt es im Bereich der Reihenfolgebildung noch die exakten Verfahren, Verfahren der künstlichen Intelligenz und die komplexen heuristischen Verfahren, Eröffnungsverfahren und Verbesserungsverfahren. Die Verfahren der künstlichen Intelligenz sind so komplex, dass sie in der folgenden Betrachtung ausgeschlossen werden. Es soll jeweils ein exaktes, ein Eröffnungs- und ein Verbesserungsverfahren betrachtet werden. Um Gemeinsamkeiten innerhalb der Verfahrensklassen identifizieren zu können, ist es notwendig, eine größere Anzahl an Algorithmen zu betrachten. So ist es möglich zu prüfen, ob sich auch komplexe Algorithmen mit SysML-Aktivitätsdiagrammen abbilden lassen und ob sich die einzelnen Teilaktionen der Algorithmen untereinander oder mit anderen Verfahrensklassen überschneiden.

Branch and Bound-Verfahren (B&B) können als exaktes Verfahren genutzt werden. Sie basieren auf Entscheidungsbäumen. Sie untersuchen sequentiell jeden Teil des Modells. Dabei ist es möglich, das zu untersuchende Kriterium frei zu wählen (Zimmermann, 2005, S. 252 ff.). Das *Branch and Bound*-Verfahren eignet sich gerade für Probleme mit kombinatorischem Charakter. Daher ist es für die Gebiete des Job-Shop- und Flow-Shop-Bereichs geeignet (Zimmermann, 2005, S. 254). Eröffnungsverfahren generieren eine Anfangslösung, während Verbesserungsverfahren gegebene Lösungen sukzessiv verbessern. Daher werden die beiden Verfahrensarten oft in Kombination ergänzend eingesetzt, um

Optimierungsprobleme näherungsweise zu lösen. Oft haben Eröffnungsverfahren die Form eines *Greedy*-Algorithmus. *Greedy* Algorithmen breiten sich an einem Startpunkt gierig aus und versuchen, durch jeden Schritt eine lokal optimale Entscheidung zu finden (Meier, 2008, S. 27). Das *Giffler-Thompson*-Verfahren (G/T) ist ein Eröffnungsverfahren, das auch für Maschinenbelegungspläne angewendet werden kann (Domschke/Scholl, 2006, S. 4). In seiner reinen Form ist es hauptsächlich für die Reihenfolgebildung bei Job-Shop-Problemen mit einmalig vorhandenen Maschinentypen geeignet (Aufenanger, 2009, S. 36). Das Verfahren bestimmt hier die jeweils frühesten Start- oder Endzeitpunkte der einplanbaren *Entities*. Sind zwei *Entities* gleichzeitig einplanbar, ist es möglich diesen Konflikt mittels ergänzender Prioritätsregeln zu lösen. Such- und Verbesserungsverfahren starten oft mit einer aus einem Eröffnungsverfahren gegebenen Lösung (Domschke und Drexl, 2006, S. 129). Es gibt Verfahren, die nur lokal nach einem Optimum suchen, und komplexere, die auch versuchen, lokale Optima zu überwinden. Dafür werden auch vorübergehende Verschlechterungen des Ergebnisses akzeptiert. Zu der komplexen Klasse der Verbesserungsverfahren gehören beispielsweise evolutionäre Verfahren, *Simulated Annealing* oder *Tabu Search*. *Tabu Search* (T/S) beruht, ähnlich wie *Simulated Annealing*, auf Nachbarschaftssuche. Es beruht ähnlich des menschlichen Gedächtnisses auf vergangenen Erlebnissen (Ergebnissen), die in einer Tabuliste gespeichert werden. Neue Ergebnisse werden iterativ ermittelt und anhand des Prinzips kleinster Verschlechterung und größter Verbesserung gespeichert. Verschlechterungen sind möglich, wenn lokal keine benachbarten Verbesserungen gefunden werden (Brüggemann, 2010, S. 47).

Es war möglich, alle Verfahren mit SysML-Aktivitätsdiagrammen abzubilden (Krög, 2011, S. 25 ff.). In Tabelle 3-16 sind alle Bausteine, aus denen die Verfahren bestehen, zusammengefasst. Es wird deutlich, dass die verschiedenen Bausteine kaum mehrfach von den komplexen Verfahren der Reihenfolgebildung genutzt werden. Interessant sind jedoch die Bausteine „Prioritätsregelverfahren“ (`use priorityRule`), in dem eine der zuvor beschriebenen Prioritätsregeln substituiert wird, und die schon vorgekommenen Bausteine „aufsteigend sortieren“ (`sort entity in queue to ascending value`) und „absteigend sortieren“ (`sort entity in queue to descending value`).

Tabelle 3-16: Zusammensetzung Komplexer Verfahren der Reihenfolgebildung (Krög, 2011, S. 66)

- trifft nicht zu	X trifft zu	O nicht exakt definierbar

Aktionsschritt	SysML-Aktion	B & B	G / T	T / S	Aktionsschritt	SysML-Aktion	B & B	G / T	T / S
Initialisierung	get initial values	x	x	x	Zug der Tabu-Liste	add characteristic of Tabu-List	-	-	x
Zielfunktion	objective function	x	-	x	Abbruch-Bedingung	termination condition	-	-	x
Verzweigungs-regel	use branching rule	x	-	-	aufsteigend sortieren	sort entity in queue to ascending value	x	-	-
Schranken-bestimmung	determine bounds	x	-	-	absteigend sortieren	sort entity in queue to descending value	x	-	-
Prioritäts-regelverfahren	use priorityRule	-	x	-	Schedule speichern	store schedule	x	x	x
Teilkriterium prüfen	part function	x	-	-					
Wert berechnen	calculate entity in queue to value	-	x	-					
Wert vergleichen	compare entities to value	-	x	-					
Generiere Nachbarschaft	generate neighbourhood	-	-	x					
Nachbar auswählen	choose best possible neighbour	-	-	x					

Lokale Verfahren der Auftragsfreigabe

Die sofortige Auftragsfreigabe gibt einen Auftrag direkt nach seiner Erzeugung frei. Die einzige abhängige Variable ist der Auftragseingang. Der Einsatz ist fast ausschließlich bei Lagerfertigung, bei der Aufträge nach ihrer Erzeugung sofort bearbeitet werden können, denkbar (Lödding, 1998, S. 298). Das Verfahren *Paired Cells Overlapping Loops of Cards with Authorisation* (Polca) ordnet die Aufträge durch andere Verfahren in einen *Schedule*, startet den Auftrag aber nur, wenn das nachfolgende Arbeitssystem über ausreichend Ressourcen verfügt (Pawellek, 2007, S. 96 ff.). Das Verfahren ist besonders für stufenübergreifende Materialflusssteuerung von Systemen mit hoher Variantenvielfalt geeignet (Rücker, 2006, S. 29). Bei der dezentralen bestandsorientierten Fertigungsregelung (DBF) wird auch erst durch ein zusätzliches Verfahren eine Liste mit Bearbeitungsstartzeitpunkten der *Entities* generiert. Zur Bearbeitung werden die *Entities* aber erst freigegeben, wenn im nachfolgenden Arbeitssystem eine Bestandsgrenze erreicht ist (Lödding, 1998, S. 423).

Neben den üblichen Initalisierungs- und Abschlussaktionen nutzen die drei lokalen Auftragsfreigabeverfahren die gemeinsame Reihenfolgebildungsaktion (`use Rule`). Weiterhin nutzen Polca und die Dezentrale Bestandsorientierte Fertigungsregelung (DBF) die komplexeren Aktionen „auf Ressourcen prüfen“ (`check succession Process for resources of entity`) und „auf Kapazität prüfen“ (`check successiuon Process for capacity of`

`entity`), welche bei den folgenden globalen Verfahren der Auftragsfreigabe wiederverwendet werden. Die Aktionen der lokalen Auftragsfreigabeverfahren zeigen, dass sie sich in verschiedenen Algorithmen wiederfinden (vgl. Tabelle 3-17).

Tabelle 3-17: Zusammensetzung Lokaler Verfahren der Auftragsfreigabe (Krög, 2011, S. 90)

- trifft nicht zu	X trifft zu	O nicht exakt definierbar

Aktionsschritt	SysML-Aktion	Sofortige Freigabe	Polca	DBF
Initialisierung	get initial values	x	x	x
Freigabe bei Erzeugung	release entity while generating order	x	-	-
auf Ressourcen prüfen	check successionProcess for resources of entity	-	x	-
auf Kapazität prüfen	check successionProcess for capacity of entity	-	-	x
Schedule speichern	store schedule	x	x	x

Globale Verfahren der Auftragsfreigabe

Die Auftragsfreigabe nach Termin gibt das *Entity* frei, wenn der geplante Starttermin erreicht wird (Dickmann, 2009, S. 180). Die statische ressourcenbedingte Auftragsfreigabe gibt die *Entities* frei, bei denen alle zur Bearbeitung geforderten Ressourcen bereit stehen. Sind mehrere Aufträge bereit, kann eine Prioritätsregel das Konkurrenzproblem regeln (Kurbel, 2005, S. 164). Die dynamische ressourcenbedingte Auftragsfreigabe gibt den Auftrag schon frei, wenn die Ressourcen noch nicht vorliegen, aber bis zum Zeitpunkt der Bearbeitung bereit stehen (Kurbel, 2005, S. 165). Beim *Constant Work in Process*-Verfahrens (CONWIP) wird wie im Kapitel 3.6.1 beschrieben ein konstanter Umlauflagerbestand sichergestellt. Daher wird das Verfahren als global angesehen. Die Reihenfolge der einzuregelnden *Entities* wird oft mit Prioritätsregeln ermittelt (Jodlbauer, 2008, S. 225 ff.). Beim *Workload-Control*-Verfahren wird ein Belastungsabgleich für die Arbeitssysteme, die zur Durchführung der Produktion vorgesehen sind, durchgeführt. Nur wenn alle Arbeitssysteme, die zur Fertigstellung des *Entity*s benötigt werden, einen vorgeschriebenen aktuellen Bestand erfüllen, wird das *Entity* freigegeben. Auch bei diesem Verfahren wird der *Schedule* der *Entities* vorher mit einem anderen Verfahren ermittelt (Lödding, 1998, S. 355.).

Besonders die Aktion der Reihenfolgenbildung durch ein anderes Verfahren (`use Rule`) oder Prioritätsregelverfahren (`use priorityRule`) wird bei dem globalen Verfahren der Auftragsfreigabe wie auch durch Verfahren anderer Klassen oft genutzt. Auch das aufsteigende Sortieren der *Entities* der Prioritätsregelverfahren (`sort entities in queue to ascending value`) wird von zwei Verfahren genutzt. Von den sich ähnelnden Aktionen der Folgeprozessprüfung auf Ressorcen oder Kapazitäten wird die Einzelprozessprüfung auf Ressourcen

(`check succession Process for resources of entity`) vom statischen Ressourcenfreigabeverfahren wiederverwendet. Es ist aber anzunehmen, dass die vier ähnelnden Aktionen von vielen Algorithmen benötigt werden. Lediglich die Aktion auf das Prüfen der CONWIP Kapazität (`check CONWIP-capacity of entity`) scheint sehr speziell zu sein und wird daher eher weniger von anderen Verfahren benötigt (vgl. Tabelle 3-18).

Tabelle 3-18: Zusammensetzung Globaler Verfahren der Auftragsfreigabe (Krög, 2011, S. 91)

	- trifft nicht zu	X trifft zu	O nicht exakt definierbar			
Aktionsschritt	**SysML-Aktion**	**Termin Freigabe**	**Statische Freigabe**	**Dyn. Freigabe**	**Conwip**	**Workload Control**
Initialisierung	get initial values	x	x	x	x	x
aufsteigend sortieren	sort entities in queue to ascending value	x	-	x	-	-
Reihenfolge-bildung	use Rule	-	x	x	-	x
Prioritäts-regelverfahren	use priorityRule	-	x	x	x	x
Folgeprozess Ressourcen	check succession Process for resources of entity	-	x	-	-	-
Folgeprozess Kapazität	check succession Process for capacity of entity	-	-	-	-	-
Folgeprozesse Ressourcen	check succession Processes for resources of entity	-	-	x	-	-
Folgeprozesse Kapazität	check succession Processes for capacity of entity	-	-	-	-	x
Conwip-Kapazität	check Conwip-capacity of entity	-	-	-	x	-
Freigabe speichern	store schedule	x	x	x	x	x

Erweiterung des Systemelements Entity

Die beschriebenen Verfahren evaluieren alle ihre Entscheidungen anhand von Attributen der *Entities*. So bestimmen die Algorithmen anhand dieser Eigenschaften die Reihenfolge der *Entities*. In Tabelle 3-19 und Abbildung 3-20 wurde das Stereotyp *Entity* um alle herausgearbeiteten Attribute zur Abbildung der betrachteten Algorithmen erweitert. Einige Attribute, die von den Verfahren benötigt werden, sind dem *Entity* zugeordneten Element-Prozess entnehmbar. Um die Redundanz zu wahren, werden sie nicht separat im *Entity* abgebildet. Beispielsweise sind die Bearbeitungszeit oder die benötigten Ressourcen dem zugeordneten Prozess des *Entitys* zu entnehmen.

Tabelle 3-19: Erweiterung des Stereotypen *Entity* zur Abbildung des Algorithmischen Modells

Attribut	Type	Syntax	Semantik
priority	Integer	0 ... 9999	Priorität des Entitys.
arrivalTime	String	Sekunde.Minute.Stunde_ Tag.Monat.Jahr	Ankunftszeit in der das Entity bereitsteht, das System zu betreten.
deadline	String	Sekunde.Minute.Stunde_ Tag.Monat.Jahr	Die Zeit,bis zu der das Entity endgültig gefertigt werden muss.
productValue	String	Euro.Cent	Steht für den Endproduktwert des jeweiligen Entitys.
partproductValue	(String) map	(Euro.Cent) Liste Produktwert/Prozess	Aktueller Produktwert des Entitys am Prozess. Gespeichert als zweidimensionale Liste Produktwert/Prozess
startTime	String	Sekunde.Minute.Stunde_ Tag.Monat.Jahr	Ein übergeordnet festgelegter Zeitpunkt an den die Bearbeitung des Entitys starten soll.

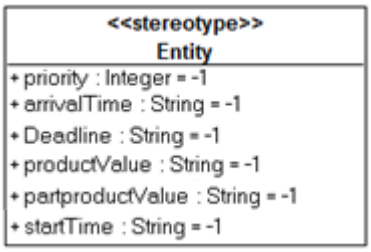


Abbildung 3-20: Erweiterung des Stereotypen *Entity* zur Abbildung des algorithmischen Modells

3.6.4 Zusammenhänge des algorithmischen Modells

Das algorithmische Modell beschreibt ein Konzept zur Einbindung, Beschreibung und Nutzung verschiedener Algorithmen im Kontext des vorgestellten Gesamtmodells. Das entwickelte Konzept kann vollständig mit SysML beschrieben werden. Es ist möglich, die Algorithmen als vorgefertigte Stereotypen im SysML-Blockdefinitionsdiagramm in das Nutzermodell einzubinden. Der Anwender kann dann vordefinierte Werte für Eigenschaften der Verfahren wählen und ihnen Systemelemente, die sie berechnen sollen, zuordnen. Weiterhin wurde ein Spektrum an Verfahren entwickelt, welches das Feld der Produktionssteuerung umfassend abdeckt und klassifiziert. Anschließend wurde das durch Literaturrecherchen erörterte Spektrum mit Algorithmen gefüllt, um diese zu untersuchen. Die gewählten Algorithmen wurden in einzelne Teile (Aktionen) zerlegt und mit SysML-Aktivitätsdiagrammen modelliert. So sollte ein Konzept erstellt werden, das es erlaubt, aus wiederverwendbaren Aktionen mit SysML-Aktivitätsdiagrammen abgewandelte und neue Algorithmen zu erstellen. Um dies technisch umzusetzen, ist es notwendig, die benötigten Sprachelemente der SysML-Verhaltensdiagramme und die einzelnen Aktionen der Algorithmen im Kontroller zu implementieren.

Insgesamt konnten etliche Aktionen von verschiedenen Verfahren genutzt werden. Besonders innerhalb der einzelnen Verfahrensgruppen, nutzen verschiedene Algorithmen spezielle Aktionen. Aber auch verfahrensgruppenübergreifend werden Aktionen wiederverwendet. Besonders bei den am Ende betrachteten globalen Auftragsfreigabeverfahren zeigten sich etliche Substitutionsmöglichkeiten. Bei einer größeren Stichprobe von Verfahren ist anzunehmen, dass die Substitutionsmöglichkeiten der Aktionen steigen. Lediglich bei den drei betrachteten komplexen Verfahren der Reihenfolgebildung zeigten sich wenige Wiederverwendungsmöglichkeiten. Es ist aber zu beachten, dass bei diesen Algorithmen lediglich einer pro Verfahrensgruppe betrachtet wurde. Insgesamt war es möglich, alle Algorithmen mit SysML-Aktivitätsdiagrammen abzubilden. Vertiefende Einblicke der Abbildung der einzelnen Verfahren mit SysML-Aktivitätsdiagrammen können einer dieser Arbeit zugrundeliegenden Masterarbeit entnommen werden (Krög, 2011).

3.7 Das Zustandsmodell

Das Zustandsmodell zeigt auf, in welchen Zuständen die Elemente des Modells sein können, und wird mit dem SysML-Zustandsdiagramm beschrieben. Es ist simulatorspezifisch, für den Entwickler von Bedeutung und für den Anwender rein informativ. Wie im Abschnitt 2.4.3 beschrieben, führt eine Transition durch ein Ereignis, das in Beziehung zu einem Verhalten steht, von einem Zustand zu einem anderen Zustand. Das bezeichnete Verhalten ist das in dieser Arbeit auf der zweiten Granularitätsstufen des Verhaltensmodells beschriebene. Da das Modellierungskonzept mit einem ereignisgesteuerten diskreten Simulator ausgeführt wird, spielen die *Events* eine gesonderte Rolle (vgl. Abschnitt 3.1). Die im Zustandsmodell gut ableitbaren *Events* sind auch die Schnittstellen vom Verhaltens- und Zustandsmodell zum Kontrollmodell. Das Zustandsmodell verbindet die Struktur mit dem Verhalten, den Zuständen und den *Events*. Interessant ist, dass die Verhaltensphasen der Verhaltensebene, die Zustände und die *Events* in einer so starken Verbindung zueinander stehen, dass sie sich voneinander ableiten lassen.

3.7.1 Zustände, Events und Verhaltensphasen

In den folgenden Ausführungen wird das in Kapitel 3.5.2 gezeigte einfache Modell der Verhaltensebene (vgl. Abbildung 3-11) in ein Zustandsmodell überführt. Es werden die Zusammenhänge der Verhaltensphasen, Zustände und *Events* und ihre Ableitungsmöglichkeiten aufgezeigt. Abbildung 3-21 zeigt, wie den einzelnen Verhaltensphasen Zustände zugeordnet werden können. Zur Bildung des Zustandsmodells sind vor allem die von den anderen Elementen angesprochenen Ressourcen und *Entities* wichtig. Sie lösen die Verhaltensphasen aus, und es ist möglich, ihnen über die Verhaltensphasen Zustände zuzusprechen.

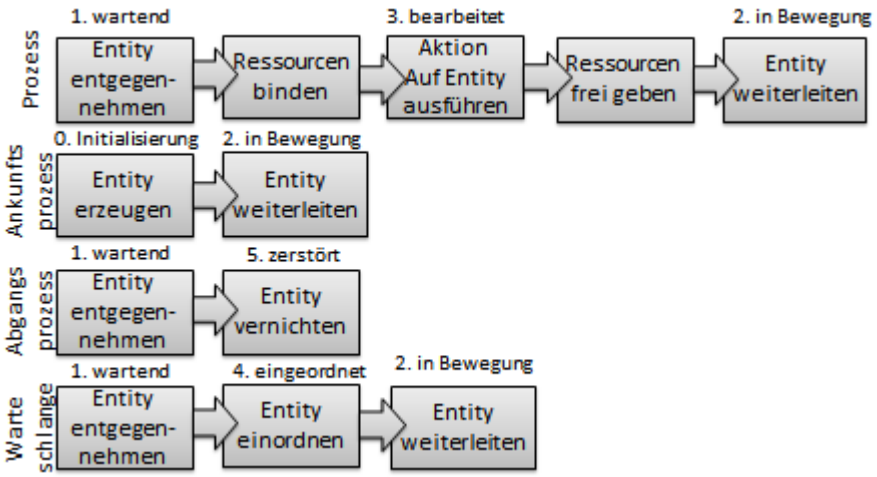


Abbildung 3-21: Zuordnung von Zuständen zu Verhaltensphasen

In Abbildung 3-22 wurde aus dem Modell der Verhaltensebene und den abgeleiteten Zuständen für das simulationsbestimmende Element *Entity* ein Zustandsdiagramm modelliert. Wie im Kapitel 2.4.3 ausgeführt, werden die Transitionen durch ein Ereignis ausgelöst und binden ein Verhalten, das zum Folgezustand führt. Der Zustandsautomat startet mit dem Pseudozustand Initialisierung, wenn ein *Entity* durch ein von der Ankunftszeit des Ankunftsprozesses ausgelöstes Ereignis in das Modell tritt. Dieses Ereignis ist mit der Verhaltensphase `erzeuge Entity` verbunden. Danach erfolgt anhand des Pseudoelementes *Decision* eine Fallunterscheidung bezüglich der Lagerkapazität des nachfolgenden Elements. Im Zustandsdiagramm wird davon ausgegangen, dass der Ankunftsprozess eine unendlich große Eingangswarteschlange hat, andernfalls wäre auch hier eine Fallunterscheidung notwendig, die das *Entity* terminieren lassen könnte. Allen Zustandsübergängen können Verhaltensphasen zugeordnet werden, einige sind auch ohne auslösenden *Event*. Der Übergang vom wartenden Zustand zur Terminierung mit dem Zustandsübergang `Entity vernichten` wird fließend vollzogen. Die Bedingung, dass das Folgeelement ein Abgangsprozess sein muss, ist kein *Event*. Bei den Übergängen von `wartend` zu `bearbeitet` und `eingeordnet`, wurde die Bedingung für die Fallunterscheidung aus Platz- und Redundanzgründen weggelassen. So wie von dem Verhaltensmuster `Entity entgegennehmen` aus Abbildung 3-21 drei verschiedene Verhaltensmuster folgen, schließen sich drei Zustände dem abgeleiteten Zustand `wartend` im Zustandsdiagramm an.

In dem Zustandsdiagramm des Elements *Entity* ist der Zusammenhang zwischen den Modellaspekten Verhaltensmuster, Zustände und *Events* gut zu sehen. Zudem zeigt es sich als eine komfortable Methode zum Ableiten der Teilaspekte.

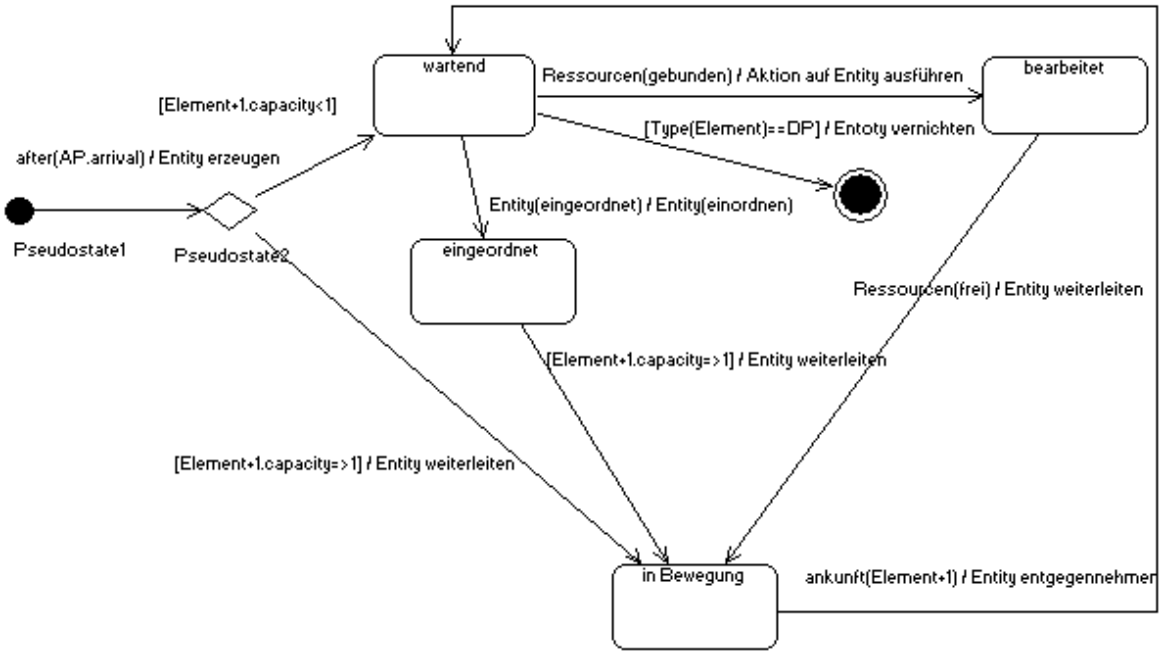


Abbildung 3-22: Zustandsdiagramm Entity

Abbildung 3-23 zeigt ein Beispiel für ein Zustandsdiagramm der Ressourcen, das aus den beiden nicht beschrifteten Verhaltensphasen `Ressource binden` und `Ressource freigeben` abgeleitet ist (Abbildung 3-23). Zusätzlich ist der Zustand `blockiert` aus der Eigenschaft `failure` der Ressource abgeleitet. Zustände können somit aus Eigenschaften des Bezugselements und Verhaltensphasen zugeordneter Elemente abgeleitet werden.

Die Zustandsdiagramme der Elemente Prozess, Ankunftsprozess, Abgangsprozess und Warteschlange werden nicht aufgeführt, da das Beispiel lediglich konzeptionelle Zusammenhänge erläutern soll. Generell sind die einzelnen Zustände, *Events* und Verhaltensphasen vom jeweiligen Simulationswerkzeug abhängig

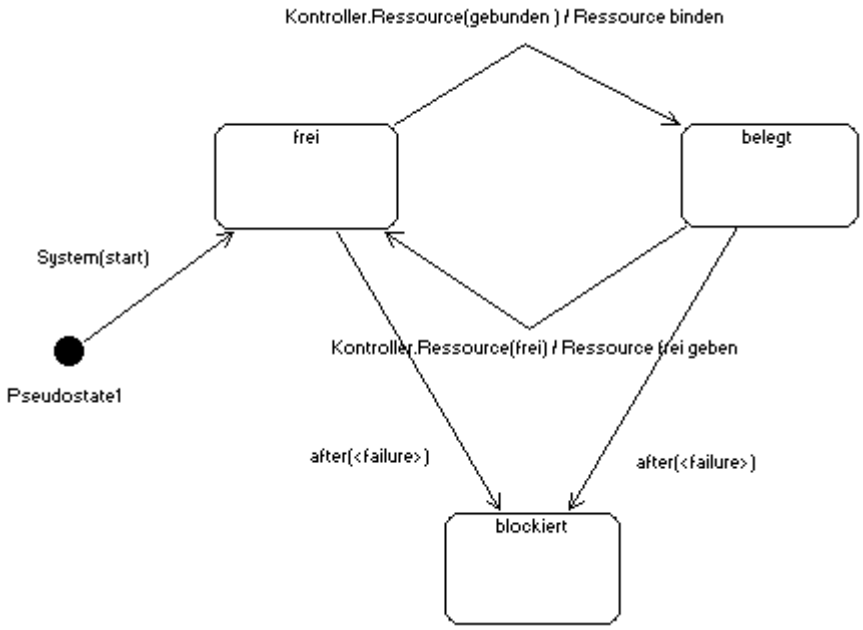


Abbildung 3-23: Zustandsdiagramm Ressource

3.7.2 Praktische Ansätze für Zustandsmodelle

Wie ausgeführt sind die Zustände der verschiedenen Simulatoren unterschiedlich. Die Simulatoren auf die möglichen Zustände zu untersuchen und diese zu sammeln, wäre nicht nutzenbringend, da die Zustände zu redundant sind. Angelidis et. al. (2012, S5 f.) beschreiben ein Zustandsmodell für Montagelinien. Abbildung 3-24 zeigt das vom American Metrics Committee standardisierte Halbleiter-Zustandsmodell für Maschinen SEMI E 10 (2001, S. 7). Jeder der in der Darstellung gezeigten Zustandszeiträume kann als Zustand verstanden werden.

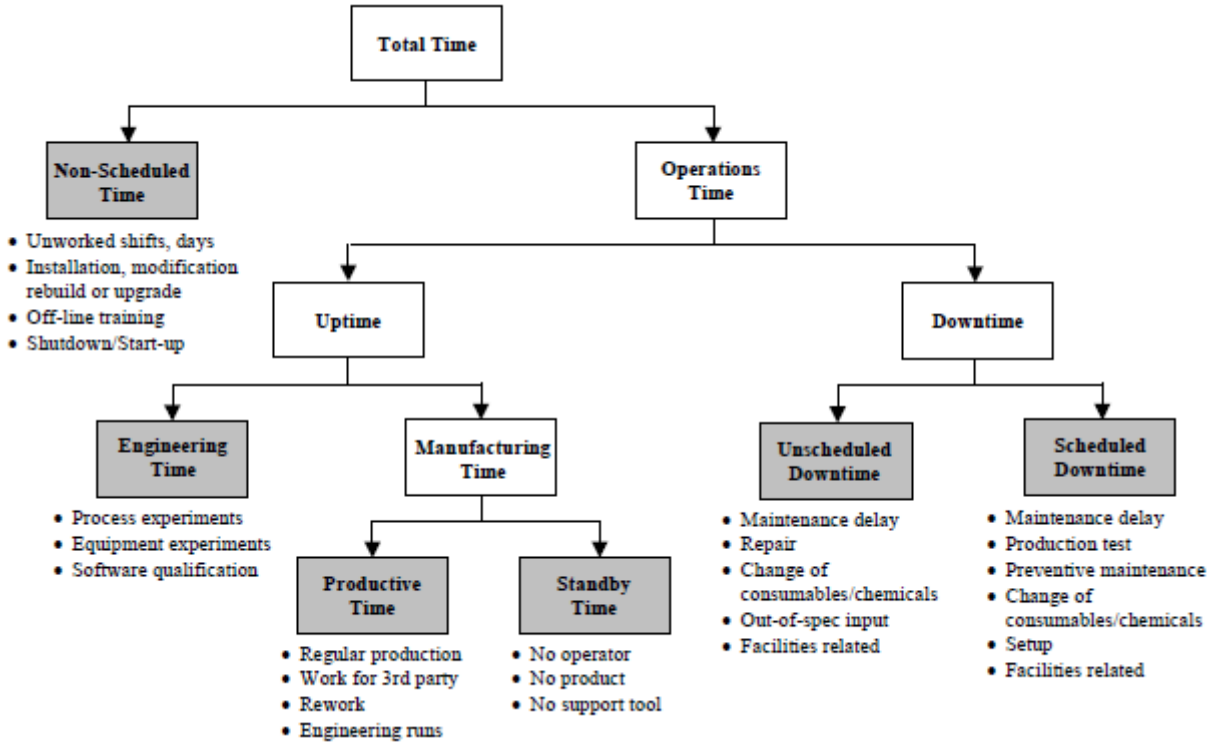


Abbildung 3-24: Beispiel für Zustände nach dem Halbleiterstandart SEMI E 10 (vgl. SEMI E 10, 2001)

3.8 Das Ereignismodell

Bei diskreten Eventsimulatoren spielen die *Events* eine wichtige Rolle. Der Ein- und Austritt jeder Verhaltensphase und jedes Kommunikationsablaufs kann Ereignisse zur Folge oder zum Anlass haben. Das Zustandsmodell zeigt, in welcher engen Beziehung Ereignisse, Zustände und Verhaltensphasen stehen. Die möglichen Ereignisse in einem Simulator sind vom Anwender nicht modifizierbar. In einer zugrunde liegenden Master Arbeit wurden verschiedene Simulatoren auch auf ihre Ereignispunkte untersucht (vgl. Kapitel 3.5.1). Dabei konnten neben vielen spezifischen auch oft genutzte Ereignisse identifiziert werden. Die vier Elemente Ankunftsprozess, Abgangsprozess, Prozess und Warteschlange haben alle beim Eingang und Abgang des *Entitys* einen Ereignispunkt. Zusätzlich hat die Warteschlange immer ein Ereignis beim Ordnen der *Entities*. Der Prozess hat zusätzliche Ereignisse beim Bearbeitungsbeginn und Bearbeitungsende sowie beim Binden und Freigeben von Ressourcen (Rehm, 2009, S. 72). Die Anzahl der Ereignispunkte kann von Simulator zu Simulator je nach seiner Komplexität stark variieren. Der Factory Explorer bietet beispielsweise 37 Ereignispunkte (vgl. Abbildung 3-25). Wie im Kapitel 3.7 gezeigt, könne die Events mit im SysML-Zustandsdiagramm dargestellt werden.

Abbildung 3-25: Ereignispunkte Factory Explorer

3.9 Das Kommunikationsmodell

Das Kommunikationsmodell beschreibt alle Kommunikationsabläufe im Modell mit Hilfe des Sequenzdiagramms. Wie in Abschnitt 3.5.2 beschrieben, können Kommunikationsabläufe im Modell den Verhaltensphasen der Verhaltensebene zugeordnet werden. Das Kommunikationsmodell ist simulatorspezifisch, für den Entwickler von Bedeutung und für den Anwender nicht modifizierbar. Im Folgenden werden für die in Abbildung 3-12 ermittelten Verhaltensphasen beispielhaft Kommunikationsmuster aufgezeigt. Dabei kommt es zur Kommunikation bei der `Ressourcen Reihenfolgebildung`, `Ressourcen Freigabe`, `Zustandsänderung`, `Reihenfolgebildung` und `Freigabe`.

Der Kommunikationsablauf der Ressourcen-Reihenfolgebildung wurde bereits in Kapitel 2.4.4 beschrieben. Beim Kommunikationsablauf zum Freigeben einer Ressource sendet der Prozess dem Kontroller eine asynchrone Nachricht, dass das Element freigegeben werden kann. Nachdem der Kontroller die Information registriert und abgestimmt hat, sendet er der Ressource eine Nachricht zu ihrer Statusänderung (vgl. Abbildung 3-26).

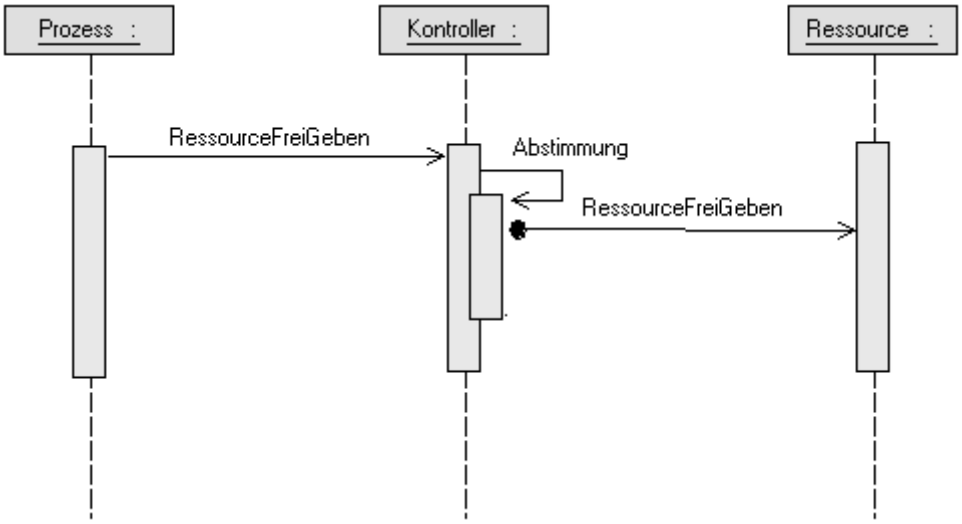


Abbildung 3-26: Kommunikationsablauf der Ressourcenfreigabe

Beim Kommunikationsablauf der Zustandsänderung sendet das *Entity* dem Kontroller eine asynchrone Meldung, die der Kontroller dann verarbeitet. Wie bei allen asynchronen Nachrichten wartet das Gegenüber (Prozess) nicht auf eine Antwort des Kontrollers (vgl. Abbildung 3-27). Vielmehr arbeitet der Prozess beim Freigeben seiner Ressourcen nach der Meldung an den Kontroller weiter, ohne auf eine Antwort zu warten. Je nach Modell könnte dieses Kommunikationsmuster auch komplexer gestaltet werden.

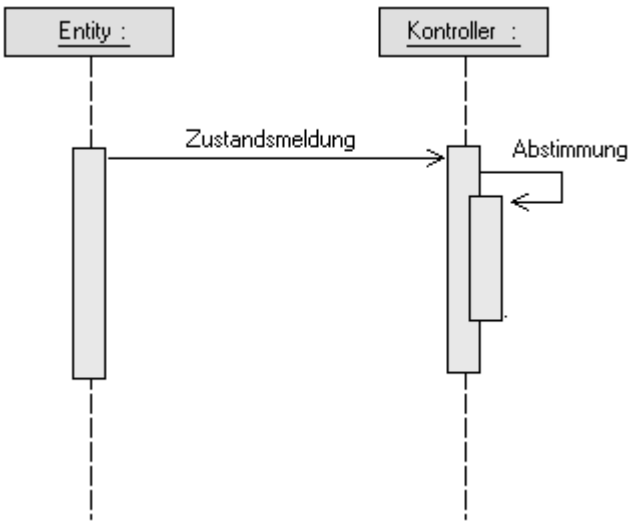


Abbildung 3-27: Kommunikationsablauf der Zustandsänderung

Die Kommunikationsmuster `Reihenfolgebildung` und `Freigabe` sind vom Ablauf der Kommunikation identisch. Während des Kommunikationsablaufs sendet das betroffene Element dem Kontroller eine Anfrage. Daraufhin berechnet der Kontroller die Anfrage und sendet dem Element dann das Ergebnis als Anweisung zurück. Die Anfrage des Elements ist synchron, daher führt es seinen Ablauf erst nach Erhalt der Antwort weiter (vgl. Abbildung 3-28).

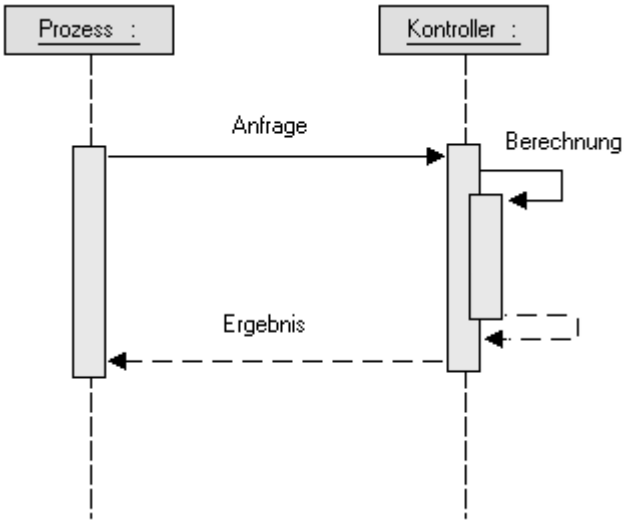


Abbildung 3-28: Kommunikationsablauf der Reihenfolgebildung und Freigabe

Abbildung 3-29 beschreibt die Kommunikation des Kontrollmodells genauer, während sie bei den anderen Kommunikationsabläufen vereinfacht wurde. Das Element aus dem Basismodell kommuniziert mit dem Interface des Kontrollmodells. Das Interface verteilt die Zustandsinformation an den Monitor und die Anfrage an den Kontroller. Der Kontroller holt sich alle Informationen, die er für die Berechnungen benötigt, vom Monitor ein, um dann die Berechnungen durchführen zu können. Abschließend gibt der Kontroller das Ergebnis als Anweisung an das Interface zurück, das dieses wiederum an das Element weiterleitet.

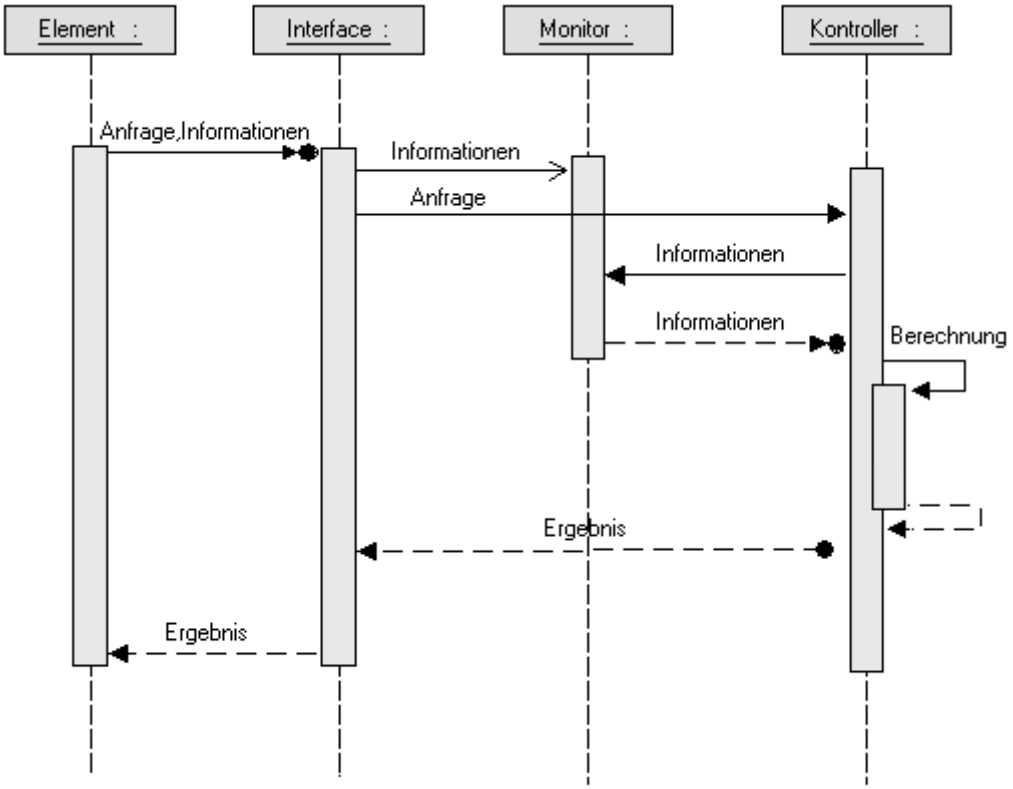


Abbildung 3-29: Kommunikationsablauf der Reihenfolgebildung und Freigabe

3.10 Das experimentelle Modell

Im experimentellen Modell definiert der Anwender die Versuchsplanung. Dafür werden Variablen für den Ablauf der Simulation festgelegt. Durch die Nutzung stochastischer Komponenten ergeben sich etliche Folgen für die Auswertung der Simulationsergebnisse (VDI Richtlinie 3633, 1997, S. 3). So sind die Ergebnisse eines Simulationslaufs mit kurzer Dauer nicht signifikant. Die Variablen zum Durchführen der Simulation unterliegen keiner festen Regel, wodurch der Anwender auf *best practices* (den besten Versuch und Erfahrungen) zurückgreifen muss. In der Literatur oft genutzte Variablen sind die Länge der Warmlaufphase, die Anzahl der Durchläufe und deren Länge (März, 2011, S. 18). Das Aufstellen des experimentellen Modells ist ein sehr komplexes Aufgabenfeld, dessen Umfang den Rahmen dieser Arbeit überschreiten würde. Kleijnen (2008) gibt einen umfangreichen Überblick und auch in der Literatur sind viele Arbeiten zum Thema zu finden (vgl.Barton, 2010; vgl. Montevechi et al., 2009). Daher sollen die genannten Variablen für einen ersten Ansatz genügen. Das experimentelle Modell kann über ein Block Experiment definiert werden.

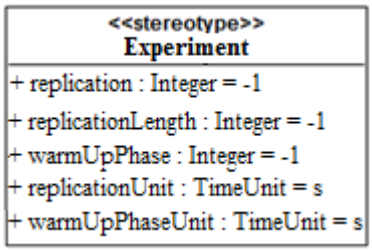


Abbildung 3-30: Block zum Abbilden des Experiments

3.11 Das Analysemodell

Im Analysemodell kann der Anwender festlegen, welche Werte er für die Auswertung der Simulation benötigt. Die Simulation berechnet diese Werte dann nach oder während ihres Laufes und gibt sie dem Anwender anschließend aus. Die möglichen statistischen Betrachtungen im Untersuchungsgegenstand sind sehr vielfältig. Die Wahl der Variablen hängt von der Möglichkeit ihrer Erfassung ab und mit stark differenten Zielen zusammen. Diese Vielfalt wird auch im von Spearman und Hopp (2011) verfassten Buch *Factory Physics* deutlich, in dem sie Zusammenhänge statistischer Variablen und deren Analyse im Bereich der Produktion beschreiben. Alle analytischen Variablen sind jedoch auf Zeitstempel der Durchlaufobjekte oder Zähler bei den Elementen eines Modells zurückzuführen, was an dieser Stelle genügen soll. Welche statistischen Größen anhand der Zeitstempel und Zählerstände im Modell ermittelt werden, bleibt dem Anwender anhand seiner Ziele überlassen. Die gewünschten statistischen Größen für die Auswertung des Modells können wie beim experimentellen Modell in Blöcken spezifiziert werden (vgl. Abbildung 3-30). Die Ausgabe der einzelnen Werte könnte beispielsweise den Blöcken des Eingangsmodells als Attribute hinzugefügt werden.

3.12 Das Gesamtmodell

Das Gesamtmodell des Untersuchungsgegenstandes wird in der Literatur bisher weniger differenziert betrachtet. Oft wird grundlegend nur zwischen Struktur und Verhalten unterschieden. „Meistens wird durch die Simulation ein Entscheidungsprozess unterstützt, bei dem mehrere Systemvarianten analysiert werden, die sich in Struktur oder Verhalten unterscheiden“ (VDI Richtlinie 3633 Blatt 1, 1997; März et al, 2011, S. 13). Die Ergebnisse dieser Arbeit zeigen auf, dass die semantische Strukturierung von Simulationsmodellen differenzierter zu betrachten ist.

Das Konzept ist strukturiert, flexibel, erweiterbar und vollkommen offengelegt gestaltet. Das Gesamtmodell besteht aus neun Teilen. Grundlegend wird einerseits zwischen dem Systemmodell und dem Simulator und andererseits zwischen dem experimentellen und dem Analysemodell unterschieden. Das Systemmodell beinhaltet alle Elemente, Verhaltensweisen, Steuerungsmechanismen, Zustän-

de, Ereignisse und Kommunikationsabläufe. Der Simulator arbeitet auf dem Systemmodell, deren Schnittstellen Ereignisse des Ereignismodells sind. Das experimentelle Modell definiert, wie der Simulator auf dem Systemmodell arbeiten soll, und das Analysemodell, welche Ergebnisse festgehalten werden sollen.

Im Modellierungskonzept werden die beiden Rollen des Anwenders und des Entwicklers unterschieden. Der Anwender nutzt das Konzept als Ingenieur, um Produktionsszenarien modellieren und optimieren zu können. Der Entwickler kann das Konzept nutzen, um Übersetzungen zu bestehenden Simulationswerkzeugen oder einen Simulator konzeptionell zu entwickeln. Bestimmte Teile des Modells sind einzig für den Aufgabenbereich des Entwicklers (beispielsweise das Zustandsmodell, das Verhaltensmodell ab der zweiten Granularitätsstufe). Dennoch ist es ein Grundprinzip des Konzeptes, die Funktionsweise auch dieser Modellteile durch SysML-Diagramme für den Anwender einsehbar zu machen. Zur Modellierung von Produktionsszenarien stehen dem Anwender das Verhaltens-, Struktur- und algorithmische Modell zur Verfügung. Auch das simulationsbestimmende experimentelle Modell und das Analysemodell sind für den Anwender von Nutzen.

Das strukturelle Modell, das Verhaltensmodell und das Zustandsmodell bilden die Grundstruktur eines Systems ab und werden als Basismodell beschrieben (vgl. Abbildung 3-31). Mit dem strukturellen Modell kann der Anwender die Struktur eines Systems mit Elementen, deren Eigenschaften und Relationen abbilden. Das Verhaltensmodell ermöglicht dem Anwender, das dynamische Verhalten der Elemente abzubilden. In der Produktion wird das Verhalten der *Entities* abgebildet, die den Maschinenpark durchlaufen. Das Verhalten kann in drei unterschiedlichen Granularitätsstufen der Prozess-, Verhaltens- und Ausführungsebene abgebildet werden. Das Zustandsmodell zeigt auf, in welchen Zuständen sich die Elemente befinden können und schafft Verbindungen zwischen den Zuständen und *Events*, welche die Zustandsübergänge einleiten und mit einem Verhaltensmuster der zweiten Granularitätsstufe einhergehen.

Das Steuerungsmodell besteht aus dem Kontrollmodell sowie dem algorithmischen Modell. Es soll ermöglichen, Algorithmen für Entscheidungen zu definieren sowie mit dem Systemmodell zu synchronisieren. Im algorithmischen Modell kann der Anwender Algorithmen für das Scheduling, Dispatching und Routing auswählen und einsetzen oder definieren. Das Kontrollmodell beschreibt die Steuerung, die die definierten Algorithmen ausführt und die Synchronisation des Steuerungsmodells mit dem Basismodell. Die Kommunikation der Steuerung und des Basismodells erfolgt an definierten Schnittstellen (Verhaltensmuster der Verhaltensebene) und wird im Kommunikationsmodell geregelt (vgl. Abbildung 3-31).

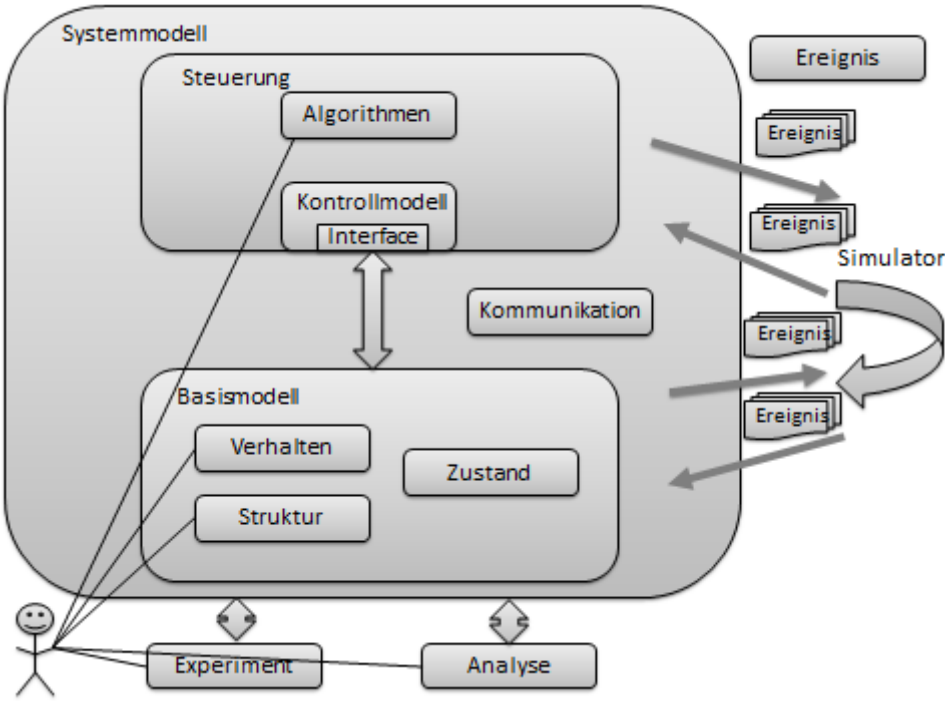


Abbildung 3-31: Das Modellierungskonzept und seine Zusammenhänge im Überblick

4 Modelltransformationen

Das vierte Kapitel widmet sich den Modelltransformationen. Im Zentrum der Dissertation steht ein Konzept für die Modellierung von Produktionsprozessen. Diese Prozesse werden oft erstellt, um sie anschließend mit bestimmten Werkzeugen wie Simulatoren oder *Scheduler* optimieren zu können. Diese Werkzeuge verlangen aber ein spezielles Input-Format. Um die mit dem Konzept dieser Arbeit modellierten Szenarien nutzen zu können, ist es daher notwendig, die Szenarien für die Zielwerkzeuge aufzubereiten. Dieser Prozess kann in das Themengebiet der Modelltransformation eingeordnet werden.

Es existieren eine Vielzahl von Publikationen zum Thema Modelltransformation, in denen verschiedene Definitionen Verwendung finden. Beispielsweise definiert die OMG die Modelltransformation „als Vorgang der Konvertierung eines Modells in ein anderes Modell des gleichen Systems“ (cf. Miller/Murkerji, 2003). Kleppe et al. 2003 geben eine Definition, die mit der Sichtweise dieser Arbeit konform ist: „*A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language*“ (vgl. Kleppe et al., 2003). Doch gibt es auch Modelltransformationen, die mehr als ein Ausgangsmodell oder Zielmodell aufweisen. Diese können zudem in verschiedenen Modellierungssprachen beschrieben sein (vgl. Mens, 2010; vgl. Biehl, 2010). Wenn die Definition von Kleppe et al. um diesen Aspekt erweitert wird, kann die folgende Definition festgelegt werden, die in einer dieser Arbeit zugrunde liegenden Masterarbeiten hergeleitet wurde (vgl. Scharfe, 2011, S. 57). Die Spezifikation der Transformationsdefinition wurde bewusst ausgelassen, um die Definition möglichst allgemein zu halten:

„Eine Modelltransformation ist die automatische Generierung eines oder mehrerer Zielmodelle aus einem oder mehreren Ausgangsmodellen, entsprechend einer Transformationsdefinition“

Zuerst erfolgt die Vorstellung von Möglichkeiten, die Modelltransformationen zu klassifizieren, um die Transformationen der Dissertation einordnen zu können. Dann werden verschiedene Methoden für die Modelltransformation aufgezeigt und im Kontext dieser Arbeit bewertet. Das Ende des Kapitels beschäftigt sich mit der ausführlichen Besprechung des gewählten Ansatzes der Dissertation.

4.1 Klassifikation von Modelltransformation

Der Begriff der Modelltransformation ist sehr weitläufig: Die Kompilierung von Java-Quellcode in Bytecode stellt ebenso eine Modelltransformation dar, wie das Übersetzen eines SysML-Modells in ein Simulationsmodell eines kommerziellen Simulators. Doch unterscheiden sich die beiden Transformationen in verschiedenen Merkmalen, wie zum Beispiel die verwendete Transformationsmethode oder die Anzahl der Zielmodelle. Mens stellte 2005 eine Reihe von Fragen zur Identifikation solcher Kriterien:

- Welche Charakteristiken einer Modelltransformation sind wichtig?
- Was soll wohin transformiert werden?
- Welche Erfolgskriterien gibt es für ein Transformationswerkzeug oder eine Transformationssprache?
- Welche Mechanismen können für die Modelltransformation verwendet werden?

Um diese Fragen zu beantworten, wurden 24 verschiedene Kriterien vorgeschlagen und beschrieben (vgl. Mens, 2005). In dieser Dissertation werden die nach Mens und Biehl vier wichtigsten Kriterien genutzt (vgl. Mens, 2010; vgl. Biehl, 2010).

4.1.1 Endogene oder exogene Transformationen

Modelle müssen in einer (Modellierungs-)Sprache beschrieben sein, um sie transformieren zu können. Wenn das Ausgangsmodell und das Zielmodell in derselben Sprache spezifiziert sind, dann wird die Transformation als endogene Transformation beschrieben. Ist ein Modell der Transformation in einer anderen Sprache beschrieben, wird die Transformation als exogene Transformation bezeichnet. Ein Beispiel für exogene Transformation wäre die Generierung von Quellcode aus einen UML-Diagramm.

4.1.2 Horizontale oder vertikale Transformation

Sind das Ausgangs- und Zielmodell einer Modelltransformation auf demselben Abstraktionslevel, wird die Klassifikation als horizontal bezeichnet, andernfalls als vertikal. Beispielsweise ist das *Refactoring*, bei dem Quellcode angepasst wird, nach dieser Definition eine horizontale Transformation. Die automatische Codegenerierung beispielsweise aus einem UML-Klassendiagramm ist nach dieser Definition eine vertikale Modelltransformation. Die Kriterien horizontal/vertikal und endogen/exogen sind orthogonal. Das heißt, dass jede Transformation in genau eine Kombination dieser Kriterien eingeordnet werden kann. Die folgenden vier Beispiele sind zur Verdeutlichung jeweils einer Kriterienkombination in Tabelle 4-1 zugeordnet worden:

- *Refactoring* bei der Umbenennung einer Klasse und aller ihr zugehöriger Referenzen.

- Formale Verfeinerung bei einer in Aussagenlogik beschriebenen Spezifikation durch das Hinzufügen neuer Axiome.
- Generierung von Java Code aus einem SysML-Blockdefinitionsdiagramm.
- Sprachmigration beim Überführen eines Java-Programms in ein C++-Programm.

Das *Refactoring* transformiert innerhalb desselben Abstraktionslevels (horizontal) innerhalb von Modellen der gleichen Sprache (endogen). Die formale Verfeinerung ist eine vertikale Transformation, da sich bei ihr das Ausgangs- vom Zielmodell hinsichtlich des Abstraktionslevels unterscheiden. Da beide Modelle in der gleichen Sprache beschrieben sind, ist die Formale Verfeinerung eine endogene Transformation. Bei der Codegenerierung sind beide Modelle auf einem unterschiedlichen Abstraktionslevel und mit einer anderen Sprache beschrieben, daher ist die Transformation exogen und vertikal. Die Sprachmigration transformiert zwischen verschiedenen Programmiersprachen (exogen), aber innerhalb eines Abstraktionslevels (vgl. Mens/Van Gorp, 2005, S. 8; vgl. Scharfe, 2011, S. 59 f.).

Tabelle 4-1 : Beispiel für die Klassifikation von Modelltransformationen

	horizontal	vertikal
endogen	Refactoring	Formale Verfeinerung
exogen	Sprachmigration	Code-Generierung

4.1.3 Deklarative oder operationale Transformationssprache

Ein weiteres Kriterium für die Unterscheidung von Transformationssprachen ist, ob eine deklarative oder eine operationale Sprache für die Modelltransformation genutzt wird. Operationale Ansätze konzentrieren sich darauf, wie transformiert werden soll. Bei deklarativen Ansätzen steht die Frage im Vordergrund, was transformiert werden soll. Dies liegt daran, dass das „wie“ bei deklarativen Ansätzen fest definiert ist. Daraus erwächst ein Vorteil für den Transformationsablauf, der bei deklarativen Sprachen ebenfalls fest definiert ist. Jedoch ist es notwendig, zu erlernen, wie diese feste Definition arbeitet und genutzt werden kann. Zudem muss ein Werkzeug vorhanden sein, welches die definierte Übersetzung einer deklarativen Übersetzung umsetzt. Der operationelle Ansatz spezifiziert den Ablauf einer Transformation unter Verwendung einer Transformationssprache. Beispiele sind die Verwendung einer dedizierten Transformationssprache oder die Umsetzung mit einer *General Purpose* Programmiersprache wie C++ oder Java. Operative Transformationen werden auch als imperative Ansätze bezeichnet. Bei deklarativen Ansätzen ist es üblich, textuelle oder visuelle Muster einzusetzen, die Relationen zwischen Ausgangs- und Zielmodellen spezifizieren. Dabei wird die Umsetzung der Transformationen durch Operationalisierung von Regeln erreicht (vgl. Cabot et al., 2010).

4.1.4 Modellmodifikation oder Modell-zu-Modell Transformation oder Modell-zu-Text Transformationen

Bei Modellmodifikationen wird kein neues Modell erzeugt, sondern das Quellmodell modifiziert. Dieser Vorgang wird häufig verwendet, um Modelle zu erweitern oder zu verändern. Ein Beispiel wäre das Hinzufügen eines neuen Zustandes in einen Zustandsautomat und die Anpassung durch neue Übergänge der restlichen Zustände. Bei der Modell-zu-Modell Transformation werden ein oder mehrere Quellmodelle in ein oder mehrere Zielmodelle überführt. Dabei können die Metamodelle unterschiedlich sein, müssen es aber nicht. Bei der Modell-zu-Text-Transformation handelt es sich oft um Codegenerierung. Konzeptionell wird aus graphischen Modellen Text abgeleitet (Czarnecki/Helsen, 2003, S. 9 ff.).

In dieser Arbeit werden SysML-Modelle in Modelle von Simulationswerkzeugen transformiert. Dabei handelt es sich um exogene, horizontale Transformationen, da die Modelle in unterschiedlichen Modellierungssprachen auf demselben Abstraktionslevel transformiert werden. Da beide Modelle mit graphischen Modellierungssprachen abgebildet sind, handelt es sich um Modell-zu-Modell Transformationen. Die gewählte Architektur teilt die gesamte Transformation jedoch in Zwischenschritte, deren genauere Betrachtung im Kapitel 4.3 erfolgt. Um die Transformationen durchzuführen, ist es möglich, operationale und deskriptive Transformationsansätze zu nutzen.

4.2 Methoden der Modelltransformation

In diesem Abschnitt werden aktuelle Modelltransformationsmethoden betrachtet und auf ihre Nutzbarkeit für diese Arbeit überprüft. Zuerst wird die von der OMG unterstützte Methode *Query View Transformation* (QVT) dargestellt, an deren Umsetzung in verschiedenen Projekten gearbeitet wird. Dann wird die imperative Umsetzung von Modelltransformationen vorgestellt, die dem operationalen Ansatz vollständig entspricht. Ein weiterer in der Forschung oft betrachteter Ansatz, sind die rein deklarativen Graphentransformationen. Abschließend wird ein Überblick über weitere Transformationssprachen wie die *Atlas Transformation Language* (ATL), *open Architecture Ware* (oAW) und *Extensible Stylesheet Language Transformations* (XSLT) gegeben.

4.2.1 Query View Transformation QVT als standardisierte deklarative oder operationale Methode

QVT ist ein Standard der OMG zur Modell-zu-Modell-Transformation, der im November 2005 veröffentlicht wurde. Transformationen mit QVT basieren auf den Metamodellen der zu transformierenden Input- und Output-Modelle. Diese müssen also vorliegen, um eine Transformation nach QVT durchzuführen. Über die Metamodelle ist es möglich, Abbildungen zu definieren. Die genutzten Metamodelle

müssen mittels der MOF beschrieben sein, um mit QVT genutzt werden zu können. Mit QVT ist es möglich, uni- und bidirektionale Modelltransformationen durchzuführen. Innerhalb einer Transformation können Modellelemente durch Zuordnungen (*traces*) konsistent erzeugt, ersetzt und gelöscht werden. Die QVT Spezifikation ist mit einem imperativen und einem deklarativen Sprachteil hybrid. Der deklarative Sprachteil besteht aus QVT-Relations und QVT-Core. QVT-Relations ist eine deklarative Sprache, die es ermöglicht, uni- und bidirektionale Relationen zwischen Modellen zu spezifizieren. QVT-Core ist eine abstrakte Relationssprache, mit der es theoretisch möglich ist, QVT-Relations vollständig abzubilden. Für den QVT Anwender ist QVT-Relations relevant aber nicht die abstrakte QVT-Core.

QVT-Operational ist eine imperative Sprache, mit der Transformationen zwischen Modellen abgebildet werden können. Die Sprache ist an existierende, imperative Programmiersprachen angelehnt und nutzt zusätzlich eine Erweiterung von OCL. Um bidirektionale Transformationen durchzuführen, muss mit QVT-Operational jede Transformationsrichtung extra implementiert werden. Zusätzlich ist es möglich, mit QVT Black Box Implementation externe Funktionen, die beispielsweise mit .Net oder Java definiert wurden, einzubinden (vgl. Abbildung 4-1).

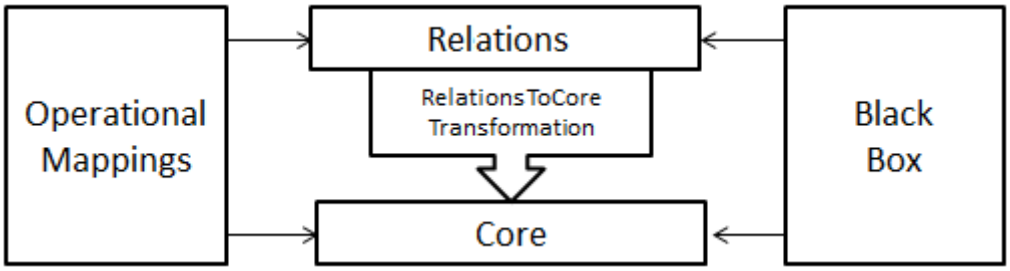


Abbildung 4-1: Architektur der Query View Transformation

Um den Standard QVT nutzen zu können, gibt es zahlreiche freie und proprietäre Werkzeuge. Beispielsweise ist das kommerzielle Werkzeug Borland Together QVT-kompatibel. Die Transformations-Engine ModelMorf implementiert QVT-Relational. Das freie Eclipse Projekt Model-To-Model (M2M) plant ein *Framework* für QVT-Operational, -Core und -Relational. Erste Testversionen sind verfügbar. Smart QVT von der France Telecom ist ein freies Eclipse-*Plugin* für QVT-Operational und medini QVT ein freies Eclipse-*Plugin* für QVT-Relational. Die Umsetzungen sind zahlreich, doch leider gibt es noch kein Projekt, das den vollen Umfang abdeckt (vgl. Tabelle 4-2).

Tabelle 4-2: QVT-Unterstüzungen verschiedener Hersteller

Name/Hersteller	QVT Operational	QVT Relational	QVT Core	Lizenz
Together Architect Boarland	X			proprietär
SmartQVT France Telecom	X			Open Source
Medini QVT IKV++ technologies AG		X*		Open Source
ModelMorf TRDDC		X**		proprietär
Tefkat University of Queensland				Open Source
Atlas Transform Language Eclipse Foundation				Open Source
Eclipse M2M Project Eclipse Foundation	In Arbeit***			Open Source

* Unterstützt keine Collection Patterns **Unvollständig *** Testversionen verfügbar

Zudem gibt es bei SmartQVT und Medini QVT seit 2009 keinen Entwicklungsfortschritt. Es scheint, als sei die Entwickler-Community der beiden Projekte nicht mehr existent. Das M2M Projekt wurde im Mai dieses Jahres in *Model-to-Model Transformation* (MMT) umbenannt und wird noch aktiv bearbeitet, ist jedoch noch nicht auf dem Entwicklungsstand, um praktische Projekte umzusetzen. Die Projekte Together Architect und ModelMorf müssen anhand ihres proprietären Ansatzes für die Nutzung dieser Arbeit ausgeschlossen werden. QVT für ATL oder Tefkat sind in ihrer Entwicklung in Bezug auf QVT auch noch nicht ausreichend vorangeschritten.

Ein Vorteil von QVT ist eine standardisierte Transformation, die relational wie auch imperativ möglich ist und auf bestehenden Standards wie MOF und OCL aufbaut. Weiterhin ist QVT herstellerunabhängig und fast universal einsetzbar. Jedoch ist der Einarbeitungsaufwand in QVT schwer abzuschätzen und wirft die Frage auf, ob der Aufwand bei komplexen praktischen Problemen überschaubar ist. Der Stand der QVT-Werkzeuge ist noch nicht ausgereift (vgl. Nolte, 2009), jedoch setzt das MMT - Projekt große Erwartungen für seine zukünftige Entwicklung. Ein weiteres Problem für die Nutzung von QVT für diese Arbeit ist, dass Ausgangs- und Zielmetamodell mittels MOF beschrieben vorliegen müssen. Zum einen sind die Zielmetamodelle der Simulatoren oft nur implizit als Benutzerhandbuch vorhanden, zum anderen müssten sie in MOF-Modelle überführt werden.

4.2.2 Imperative Transformation

Im Rahmen dieser Arbeit wurde eine Systemarchitektur entwickelt, die aus SysML-Modellen automatisch Modelle für Simulationswerkzeuge generiert. Um den Ablauf möglichst effektiv zu gestalten, wurde eine mehrschichtige Architektur gewählt (vgl. Abbildung 4-2). Zuerst muss das SysML-Modell mit Hilfe eines Modellierungstools entworfen werden, welches das Modell in einem zur Weiterverarbeitung geeigneten Austauschformat bereitstellt.

Wenn das SysML-Modell in einem geeigneten Austauschformat bereit steht, ist es möglich, das Modell in sein Äquivalent für ein Simulationsmodellierungswerkzeug zu transformieren. Da es möglich sein soll, ein in der SysML vorliegendes Modell für verschiedene Simulationsprogramme transformieren zu können, muss für jedes Programm eine eigene Ausgabe erzeugt werden. So erhält jedes Programm ein für sich geeignetes Modell in speziellem Format. Um den Arbeitsaufwand dafür möglichst gering zu halten, wurde die Erzeugung des Modells in zwei Schritte geteilt (vgl. Abbildung 4-2). Im ersten Schritt kommt ein Programm, das als *Parser* bezeichnet wird, zum Einsatz. Dieses liest das im Austauschformat gespeicherte SysML-Modell ein und filtert alle wichtigen Informationen, die es anschließend in das interne Modell überführt. Im zweiten Schritt werden *Translator-Plugins* eingesetzt, die das interne Modell jeweils für ein Simulationsprogramm übersetzen. Dafür filtert ein *Translator-Plugin* alle für das Simulationsprogramm wichtigen Daten aus dem internen Modell und bringt sie in das spezielle Format des Simulators. Da jeder Simulator ein eigenes Format hat, muss jeweils ein eigenes *Translator-Plugin* geschrieben werden.

Im ersten Schritt wird das SysML-Modell in ein internes Java-Modell eins zu eins als horizontale Transformation übertragen. Der erste Schritt ist exogen, da von SysML auf Java übertragen wird. Die Programmierung des Transformationsschrittes wurde imperativ mit einem Java-*Parser* durchgeführt. Im zweiten Teilschritt wird mit einem *Translator-Plugin* das in Java hinterlegte Modell in das Format eines Simulationswerkzeuges übertragen. Da die Ausgangs- und Zielsprachen wieder unterschiedlich sind, ist auch dieser Transformationsschritt exogen. Der Abstraktionslevel des Zielmodells hängt vom jeweiligen Simulationswerkzeug ab. Sämtliche in der Arbeit umgesetzten Transformationen wiesen einen identischen Abstraktionslevel auf, waren also horizontal. Auch der zweite Transformationsschritt wurde mit einer imperativen Sprache umgesetzt. Ein Vorteil der imperativen Sprache besteht im niedrigen Einarbeitungsaufwand und der hohen Gebräuchlichkeit. Zudem sind deklarative Ansätze, wie in diesem Kapitel beschrieben, noch nicht ausreichend ausgereift.

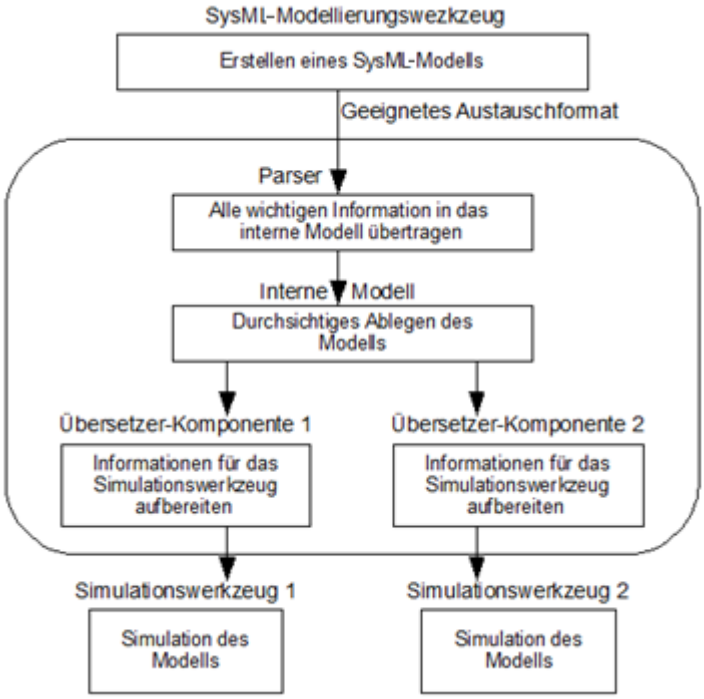


Abbildung 4-2: Systemarchitektur des Transformationssystems

4.2.3 Deklarative Transformation durch Graphgrammatiken

Eine Methode für die deklarative Modelltransformation sind Graphenersetzungen. Modelle können zum großen Teil als Graphen verstanden werden, die sich mit speziellen Werkzeugen darstellen und mit Ersetzungsregeln transformieren lassen. Eine Graphgrammatik besteht aus verschiedenen Regeln (Graphproduktionen), die jeweils einen Graph auf der linken Seite *Left Hand Side* (LHS) und einen Graph auf der rechten Seite *Right Hand Side* (RHS) besitzen. Zudem existiert ein Ausgangsgraph, der transformiert werden soll. Wenn ein Schema im Graph auf die LHS passt (*match*), wird eine Regel angewendet. Wurde ein solches Muster gefunden, wird es durch die RHS der jeweiligen Transformationsregel ersetzt. Die Regeln werden solange angewendet, bis kein Match mehr zutrifft. Passt mehr als ein Match auf eine Regel, wird eine Regel zufällig ausgewählt. Die Reihenfolge der Regelanwendung ist nicht festgelegt, somit sind Graphgrammatiken nicht-deterministisch (vgl. Cabot et al., 2010).

Beschreibung von PGGs und TGGs

Es gibt spezielle Ausprägungen von Graphgrammatiken, wie die *Pair Graph Grammars* (PGG) (vgl. Leitner, 2006) oder die *Triple Graph Grammars* (TGG) (vgl. Grunske et al., 2007), die sich für die Modeltransformation besonders gut eignen. PGGs bestehen aus Tupeln der Form `((S,T),m)`, wobei `S` und `T` Graphproduktionen darstellen, die auf zwei verschiedenen Graphen arbeiten. Die Relation `m` ordnet Knoten der linken Produktion, Knoten der rechten zu. So beschreibt ein *Pair Graph Grammars* (PGG) die parallele Entwicklung von zwei Modellen. Wenn ein Modell auf der linken Seite konstruiert wird, kann mit derselben Sequenz ein Modell auf der rechten Seite konstruiert werden. Abbildung

4-3 zeigt ein Beispiel für eine PGG Regel, die aus einer linken und rechten Graphenproduktion sowie einer Relation m besteht.

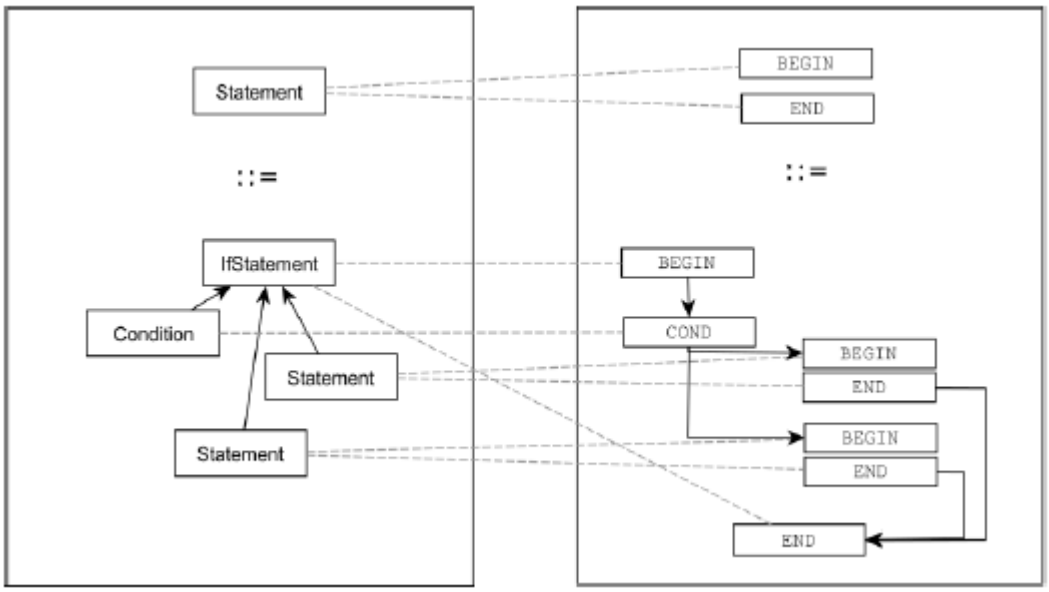


Abbildung 4-3: Beispiel einer PGG (vgl. Leitner, 2006; Scharfe, 2011, S. 62)

TGGs erweitern das Konzept der PGGs um einen Korrespondenzgraphen der mit der linken und rechten Seite eines Graphen verknüpft ist. Über den Korrespondenzgraphen ist es möglich, Informationen der Transformation und über deren Ablauf zu speichern. Diese können beispielsweise für die Überprüfung der im Kapitel 5.1 beschriebenen Konfluenz verwendet werden (vgl. Hermann, 2010). Abbildung 4-4 zeigt ein Beispiel für eine TGG.

Zur Anwendung einer Modelltransformation können Vorwärtsregeln, Rückwärtsregeln und Konsistenzregeln unterschieden werden (vgl. Cabot et al., 2010). Bei einem Ausgangsmodell der Sprache A und einem Zielmodell der Sprache B kann die Transformation A → B durch eine Vorwärtsregel und die Transformation B → A durch eine Rückwärtsregel beschrieben werden. Zudem ist es möglich, Konsistenzregeln zu definieren, die den Korrespondenzgraphen nutzen.

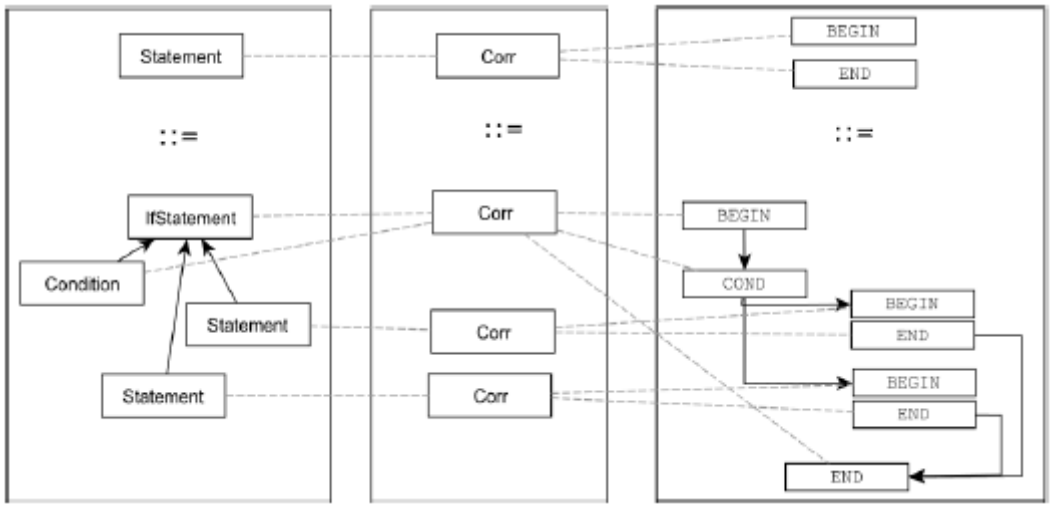


Abbildung 4-4: Beispiel einer TGG (vgl. Leitner, 2006, S. 16; vgl. Scharfe, 2011, S. 62)

Praktische Umsetzungen für die Transformation mit TGGs

Für die Umsetzung der Transformation von TGGs existieren zwei unterschiedliche Ansätze (vgl. Greenyer et al., 2008; vgl. Kindler/Wagner, 2007). Beim generativen Ansatz werden die TGG Regeln in Quellcode übersetzt. Mit dem resultierenden Programm ist es dann möglich, Transformationen umzusetzen. Beim interpretierenden Ansatz werden die Regeln der TGG während der Transformation ausgewertet. Für beide Ansätze stehen verschiedene Werkzeuge zur Verfügung.

Beispielsweise ist die *Fujaba Tool Suite* ein gebräuchliches Werkzeug für den generativen Ansatz. Das Programm enthält einen Editor zum graphischen Erstellen von TGGs und Komponenten für die Generierung des Transformationsprogramms in Java. Zuerst müssen die Metamodelle des Ausgangs- und Zielmodells mit UML Klassendiagrammen erstellt werden, um dann Transformationsregeln in einem TGG Diagramm zu spezifizieren. Mit dem daraus exportierten resultierenden Java Programm ist es möglich, Modelle im *Extensible Markup Language* (XML) *Metadata Interchange* (XMI) Format einzulesen und zu transformieren.

Die Universität Paderborn hat zur Umsetzung des interpretierenden Ansatzes den TGG-Interpreter entwickelt, der auf dem *Eclipse Modeling Framework* basiert. Die Metamodelle des Ziel- und Ausgangsmodells können auf dem *Ecore*-Modell spezifiziert werden, was entweder mit dem *Ecore* Diagramm oder *Ecore*-Standardeditor durchgeführt wird. Im Anschluss ist es möglich, die TGG-Regeln visuell zu modellieren. Nun ist die Spezifikation der Transformation fertiggestellt, und es können Transformationseinstellungen für den TGG-Interpreter konfiguriert werden. Dann erfolgt die Eingabe der Ausgangsmodelle im XMI-Format, um von TGG-Interpreter in ein XMI-Zielmodell umgewandelt zu werden.

Praktische Betrachtungen haben gezeigt, dass die Beschreibung mit Graphen vor allem bei großen und komplexen Problemen nur sehr schwer möglich ist. Die Modelle werden rasch unübersichtlich und

schwer verständlich. Auch der Einarbeitungsaufwand ist als hoch einzuschätzen, da Erfahrungen im Bereich der Graphentransformation oft nicht vorhanden sind. Zudem wird für die Umsetzung ein passendes Werkzeug benötigt. Während die Metamodelle in *Fujaba* mit UML-Klassendiagrammen abgebildet werden, bietet der TGG-Interpreter es an, die Metamodelle mit dem mächtigeren *Ecore*-Modell zu spezifizieren. Dagegen ist der Ansatz des TGG-Interpreters weniger performant als der generative Ansatz, was sich vor allem bei größeren Modellen bemerkbar macht. Selbst wenn ein passendes Werkzeug zur Verfügung stehen würde, wäre die nicht ausreichende Beschreibung der Zielmodelle auch bei Graphentransformation ein Problem (vgl. Greenyer et al., 2008).

4.2.4 Weitere Ansätze für die Modelltransformation (ATL, oAW und XSLT)

Die ATL ist eine hybride Programmiersprache zum Transformieren von Modellen. Sie vereint Konzepte aus imperativen und deklarativen Programmiersprachen. ATL wird von ATLAS INRIA & LINA entwickelt und von der OMG in Auftrag gegeben. Das Werkzeug steht als Eclipse-*Plugin* mit einem eigenen Debugger und Editor zur Verfügung und ist kostenfrei. Die Transformation besteht aus den drei Teilen *Header*, *Helper*-Operation und Regeln. Im *Header* werden der Transformationsname, das Quellmetamodell und das Zielmetamodel, sowie der Import von Bibliotheken spezifiziert. *Helper*-Operationen sind im Wesentlichen Methoden, die anderen imperativen Programmiersprachen ähneln und zur Unterstützung der Transformation eingesetzt werden. Die ATL Regeln beschreiben, wie aus dem Ausgangsmodell das Zielmodell erzeugt wird. Neben dem Einarbeitungsaufwand bleibt jedoch offen, inwieweit die Vorteile des deklarativen Ansatzes durch die hybride Programmiersprache erhalten bleiben (vgl. Jouault et al., 2008).

OpenArchitectureWare (oAW) ist ein Werkzeug für die deklarative Modelltransformation. Bis 2003 war oAW ein kommerzielles Werkzeug, seitdem ist es *Open Source*. oAW kann in Eclipse integriert werden und enthält einen deklarativen Transformationsansatz. Durch eine große Anzahl an Sponsoren (beispielsweise Informatig AG, b+m) und einer großen Online-Community ist das Werkzeug weit entwickelt. Es besteht aus einem *Parser*, einem Analyzer, einem Transformer und einem Generator. Die Komponenten werden von einer *Workflowengine* aufgerufen, die sich durch eine Konfigurationsdatei einstellen lässt. Es werden einige Formate für das Parsen, darunter auch der standardisierte UML 2.0 und SysML-*Output,* unterstützt. Um nicht unterstützte Metamodelle einzubinden, ist es möglich, eigene *Parser* Komponenten zu implementieren. OAW benötigt für die Model-zu-Model-Transformation einen nicht zu unterschätzenden Einarbeitungsaufwand (vgl. Schuster, 2010).

Die *Extensible Stylesheet Language* XSLT wird für Transformationen von XML-Dokumenten genutzt und dafür auch vom World Wide Web Consortium (W3C) empfohlen. Die Sprache baut auf die Baumstruktur von XML-Dokumenten auf und nutzt Umwandlungsregeln für die Transformation. Die Regeln werden in einem *Stylesheet* beschrieben und von einer speziellen Software (so genannter XSLT-Prozessor) genutzt, um ein Ausgangsmodell in ein Zielmodell zu überführen. Die Sprache wie

auch die meisten Prozessoren sind frei. Neben dem Einarbeitungsaufwand und dem Level an Abstraktion ist zu bemängeln, dass nicht alle Austauschformate der Zielmodelle in dieser Arbeit in XML vorliegen (vgl. Mangano, 2006).

4.3 Ein praktischer Ansatz für die Modelltransformation

Nachdem der gewählte Transformationsansatz vorgestellt wurde (vgl. Abschnitt 4.2.2), ist es möglich, diesen in den folgenden Betrachtungen genau zu spezifizieren. Der erste Abschnitt legt sein Augenmerk auf den Ansatz, seine einzelnen Bearbeitungsschritte und Komponenten. Der zweite Abschnitt beschäftigt sich mit dem *Parser*, der den ersten von zwei Transformationsschritten ausführt. Neben der Transformation von SysML in die Modellbeschreibung eines Simulationswerkzeugs, ist es auch möglich Modelle von einem Simulationswerkzeug nach SysML zu überführen. Die dazu benötigte Systemarchitektur und ihre Besonderheiten werden im dritten Abschnitt ausgeführt.

4.3.1 Konkretisierung des genutzten Modelltransformationsansatzes

Im ersten Schritt wird mit einem geeigneten *Modellierungstool* ein SysML-Modell erstellt. Wie einleitend erwähnt, muss das *Modellierungstool* ein geeignetes Austauschformat bereitstellen, alle im Modellierungskonzept erarbeiteten SysML-Modellierungselemente anbieten und zur Erstellung von großen Modellen geeignet sein. Als Austauschformat bietet sich XMI als Standard der OMG zum Austausch objektorientierter Modelle zwischen Softwareentwicklungswerkzeugen an (vgl. OMG, 2009). Ein großer Vorteil ist, dass XMI auf das wohlstrukturierte, hierarchische Datenformat XML aufbaut, durch dessen Eigenschaften sich Daten leichter verarbeiten lassen. Zudem zeigt sich aus den beiden durchgeführten Marktuntersuchungen XML als das am häufigsten angebotene Austauschformat der Simulationswerkzeuge. Das Modellierungswerkzeug MagicDraw mit SysML-*Plugin* bietet XMI als Austauschformat an und wird zusätzlich von Tim Weilkiens für die Entwicklung großer SysML-Modelle genutzt (vgl. Schulze, 2008). Anhand dieser Eigenschaften wurde MagicDraw im ersten Ansatz als Modellierungstool für das praktische System gewählt. Da Systemingenieure spezielle Anforderungen an ein Modellierungswerkzeug stellen, wurde in dieser Arbeit ein Modellierungswerkzeug entwickelt, das an die speziellen Anforderungen angepasst wurde (vgl. Kapitel 6). Das Werkzeug baut auf das *Opensource* SysML-Modellierungswerkzeug TOPCASED auf.

Im zweiten Schritt wird das erstellte SysML-Modell mit einem als *Parser* bezeichneten Programm in das interne Modell übertragen. Dabei führt der *Parser* zwei Schritte aus: Zuerst liest er alle wichtigen Daten aus dem Austauschformat, um sie anschließend in das interne Modell zu übertragen. Der *Parser* wurde in einer zugrundeliegenden Arbeit entwickelt (vgl. Partzsch, 2010). Der *Parser* überträgt die mit dem SysML-Modellierungstool entworfenen Modelle in das interne Modell. Dabei erkennt er ihm vorgegebene syntaktische Elemente. In SysML können Sachverhalte oft mit verschiedenen Modellie-

rungselementen dargestellt werden. So ist es möglich, Relationen mit Assoziationen der Blöcke, Internen Blockdiagrammen oder *Part Properties* darzustellen (vgl. Kapitel 2). Eine Relation setzt zwei Elemente in Beziehung, wodurch sie beispielsweise Ressourcen oder Modi zu Prozessen zuordnen kann. Dem Ingenieur sollen zur Modellierung der Sachverhalte alle SysML-Relationen gestattet werden, da die Modellierung durch vordefinierte Einschränkungen eher komplizierter wird. Daher war es wichtig, den *Parser* möglichst flexibel zu entwickeln, so dass er mit verschiedenen Modellierungsmöglichkeiten umgehen und bei Erweiterungen des Modellierungskonzepts einfach angepasst werden kann. Da MagicDraw und TOPCASED beide auf das XMI Datenformat EMF UML 2.0 aufbauen, war es möglich, den *Parser* ohne erheblich erhöhten Arbeitsaufwand für beide Programme zu entwickeln.

Beim Erstellen des SysML-Modells ist es wichtig, darauf zu achten, dass es bestimmten Strukturen genügt, die der *Parser* erkennen kann. Diese Strukturen werden in einem Metamodell festgelegt. Das erstellte Modell muss sich vollständig dem Metamodell zuordnen lassen, jedes Objekt und jede Eigenschaft müssen auf das Metamodell abgebildet (*gemappt*) werden können.

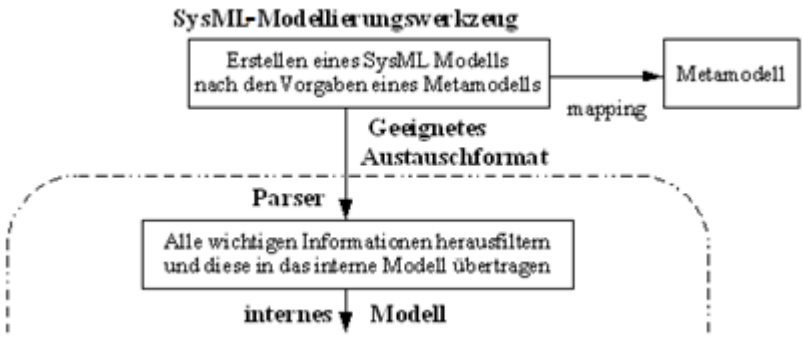


Abbildung 4-5: Erstellen eines SysML-Modells nach den Vorgaben des Metamodells

Die in Abbildung 4-2 gezeigte Architektur gibt dem internen Modell eine besondere Rolle, da es für die Entwicklung aller *Translator-Plugins* den Ausgangspunkt darstellt. Es muss alle Informationen enthalten und trotzdem gut überschaubar sein. Die Transformationen wurden zu einem Zeitpunkt entwickelt, an dem das Modellierungskonzept auf das Verhaltensmodell und das strukturelle Modell beschränkt war. Daher ist das interne Modell eine Java-Klassenstruktur, die auf dem Metamodell des strukturellen Modells (vgl. Abbildung 2-2) und einem Ausschnitt des SysML-Metamodells für Aktivitätsdiagramme (vgl. Abbildung 3-8) basiert.

Nachdem ein SysML-Modell in das interne Modell übertragen wurde, ist es möglich, es für verschiedene Simulationstools aufzubereiten. Da jedes Simulationswerkzeug eine eigene Syntax und Semantik sowie ein eigenes Datenaustausformat nutzt, muss für jedes Simulationswerkzeug ein eigenes *Translator-Pugin* entwickelt werden (vgl. Abbildung 4-6).

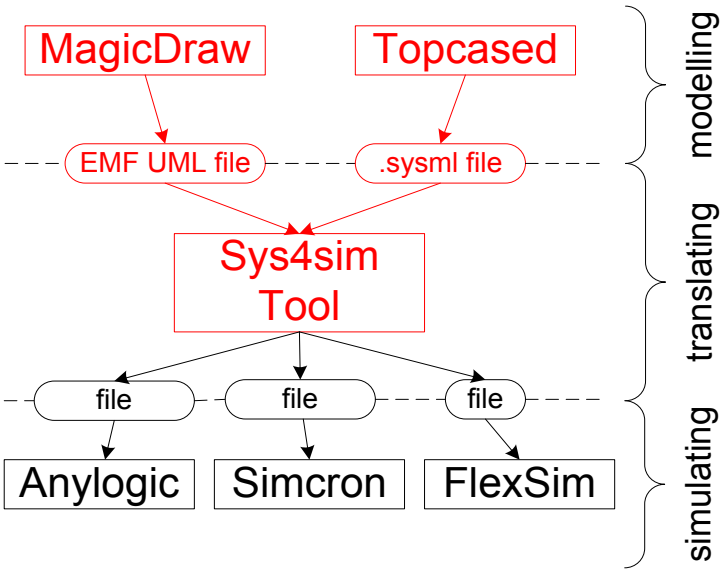


Abbildung 4-6: konkretisierte Systemarchitektur des Transformationssystems

Ein Vorteil der gezeigten Architektur ist, dass das SysML-Modell mit möglichst geringem Aufwand vom internen Modell in verschiedene Simulationsmodelle überführt werden kann. Der Nachteil ist eine größere Fehleranfälligkeit, die durch die zwei Transformationsschritte entsteht. Da die erste Transformation jedoch auf demselben Metamodell beruht, ist die Fehleranfälligkeit im ersten Transformationsschritt sehr gering. Die *Translator-Plugin*s wurden von Studenten der TU-Dresden in ihren Abschlussarbeiten entwickelt. Da das Informatikstudium an der TU Dresden eine eingehende Beschäftigung mit der Programmiersprache Java vorsieht, konnte das interne Java-Modell den Einarbeitungsaufwand der Studenten stark verringern.

Wie ausgeführt, sind der *Parser* und das interne Modell aktuell auf dem Stand des Verhaltens- und strukturellen Modells. Zudem wurden *Translator-Plugins* für die Programme Anylogic, Flexsim, Simcron und Factory Explorer entwickelt, die in den folgenden Ausführungen betrachtet werden. Auch die *Translator-Plugins* wurden für das Verhaltensmodell und das strukturelle Modell entwickelt. Da die Transformationen auf verschiedene Metamodelle abbilden, sind die semantischen Eigenheiten der Zielmodelle und die Möglichkeiten ihrer Überwindung besonders interessant. Bei der Auswertung der verschiedenen Transformationen zeigt sich die Allgemeingültigkeit des Metamodells dieser Arbeit.

4.3.2 Architektur des *Parsers*

Da in der Arbeit das auf dem XML basierende Format XMI verwendet wird, wurden XML kompatible *Parser* betrachtet. Es gibt eventbasierte (beispielsweise die *Simple API for XML* (SAX)) und baumbasierte (beispielsweise *Document Object Model* (DOM)) XML *Parser*. Baumbasierte *Parser* lesen das vollständige XML-File ein. Der Aufbau des Baumes benötigt wesentlich mehr Speicher als das Originalfile, und der Prozess des Einlesens eine gewisse Zeit. Ereignisbasierte Anwendungen betrachten

immer nur einzelne Ereignisse, bevor sie zum nächsten gehen. Obwohl die ereignisbasierten Anwendungen so wesentlich schneller sind und weniger Ressourcen verbrauchen, haben sie kein Wissen über den vollständigen Baum und können nur die aktuellen lokalen Daten behandeln (vgl. Harold, 2003). Wegen der besseren Skalierbarkeit bieten sich die ereignisbasierten *Parser* eher für große Projekte an, während die baumbasierten komfortabler sind. Da die Größe der betrachteten Modelle erheblich sein kann, wurde den ereignisbasierten *Parsern* der Vorzug gegeben.

Im Bereich der ereignisbasierten *Parser* haben sich vor allem SAX durchgesetzt (vgl. SAX, 2010). Harold vergleicht die beiden *Parser* und favorisiert den *pull*-basierten *Streaming API for XML* (StAX) *Parser* gegenüber den *push*-basierten SAX *Parser* (vgl. Harold, 2003). Den größten Vorteil sieht Harold darin, dass die *pull*-basierten Anwendungen auf dem *Iterator*-Entwurfsmuster basieren. Der *Iterator* bietet Möglichkeiten an, andere Klassen zu durchqueren. Dadurch muss der Entwickler nicht angeben, wie der *Parser* zum nächsten Ereignis gelangt. Der Code kann so leichter umgesetzt und verstanden werden. Die typische Architektur eines StAX *Parsers* arbeitet mit einer `while-loop` Schleife, die von Ereignis zu Ereignis springt und diese mit `switch-case` Statements unterscheidet. Der SAX *Parser* arbeitet mit Polymorphismen, die schwerer zu verstehen und langsamer sind (vgl. Harold, 2003). In einer ersten Version wurde für die Umsetzung dieser Arbeit ein SAX-*Parser* genutzt (vgl. Lange, 2008). Nach gründlichen Recherchen und den gerade beschriebenen Ergebnissen wurde in einer zweiten Version einen StAX-*Parser* ausgewählt (vgl. Partzsch, 2010). Der direkte Vergleich bestätigt die beschriebenen Erkenntnisse.

Die modellrelevanten Informationen sind über die XMI-Datei verteilt. Beispielsweise sind Blöcke, Stereotypen und Assoziationen in verschiedenen Bereichen der XML-Datei hinterlegt. Wie einführend erwähnt, ist es bei ereignisbasierten *Parsern* ein Problem, dass kein Wissen über Ereignisse besteht, die sich außerhalb der lokalen Betrachtungen befinden. So ergab sich die Schwierigkeit, dass Schritte Informationen benötigen, die noch nicht vorhanden sind. Daher ist es notwendig, den Prozess des Parsens in verschiedene Schritte zu unterteilen (vgl. Abbildung 4-7). Nachdem das Modell als XML-File exportiert wurde, werden vom StAX-*Parser* zuerst die grundlegenden Elemente geparst. Anschließend werden die Informationen in das interne Java-Modell separiert und abschließend verlinkt.

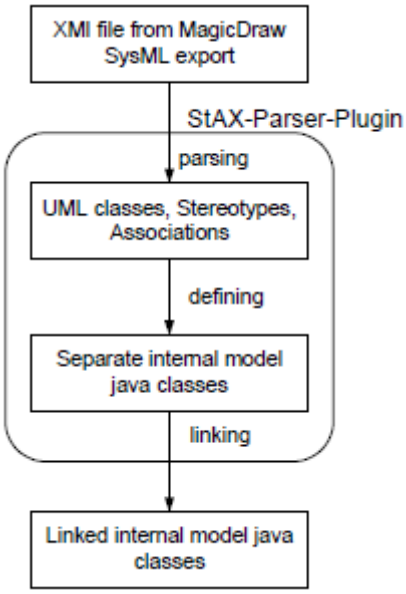


Abbildung 4-7: Architektur des *Parser* (vgl. Partzsch, 2010, S. 15)

4.3.3 Die Rückwärtstransformation vom Modellierungswerkzeug

Im Bereich der Simulation von Produktionsszenarien gibt es in der Industrie sehr große Modelle. Beispielsweise wurde für ein Simulationsprojekt in der Dresdner Betriebsstätte der Infineon AG ein Modell mit einer Größe von ca. zehn Mannjahren Arbeitsaufwand entwickelt (vgl. Noack, 2012). In der Praxis werden solche Modelle von Firmen teilweise mit kommerziellen Simulationswerkzeugen entwickelt. Ein Wechsel zu anderen Werkzeugen ist durch die Größe des Modells kaum durchführbar. Die Transformation von einem Simulationswerkzeug in das interne Modell würde es erlauben, das Modell für ein anderes Simulationswerkzeug oder ein SysML-Modellierungswerkzeug aufzubereiten.

Mit einigen Änderungen der Systemarchitektur ist es möglich, die Grundlage für ein System, das aus Modellen von Simulationswerkzeugen SysML-Modelle generiert, zu schaffen. So können die Modelle in beide Richtungen beliebig transformiert werden. Während das Metamodell sowie das interne Modell beibehalten werden können, ist es notwendig, den *Parser* und die *Translator-Plugins* neu zu entwickeln. Die *Translator-Plugins* übertragen die Modelle der Simulatoren in das interne Modell und der *Parser* bereitet die Informationen nun für das SysML-Modellierungswerkzeug auf (vgl. Abbildung 4-8). Die Transformation des *Parsers* bleibt exogen und horizontal – genauso wie die Transformationen der *Translator-Plugins*. Die imperative Übersetzung wird bei allen Komponenten beibehalten.

Wie auch bei den anderen Transformationen unterliegt die Übersetzung den Grenzen des Metamodells. Sämtliche in den Simulationswerkzeugen entwickelte zu übersetzende Modelle müssen mit dem Metamodell des internen Modells abbildbar und vom Entwickler im *Parser* berücksichtigt werden. Die vollständige Auswahl der im Simulationswerkzeug angebotenen Elemente ist daher mit begrenztem Aufwand nicht umzusetzen.

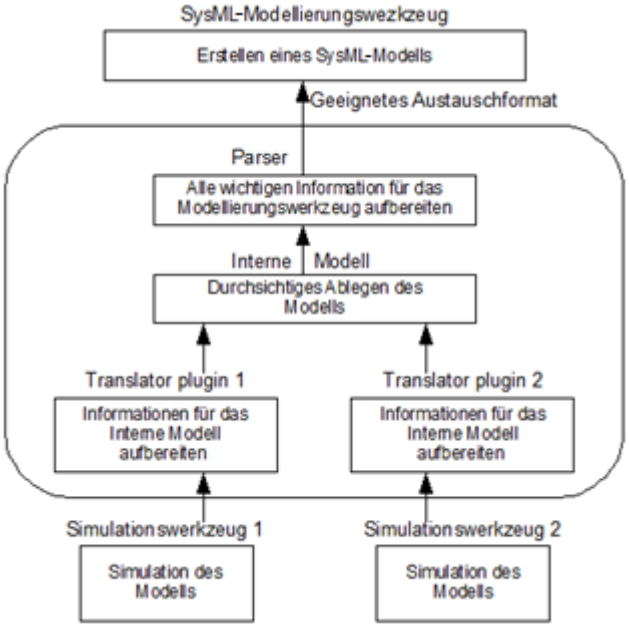


Abbildung 4-8: Systemarchitektur für das Einlesen von Modellen

4.4 Modelltransformation von SysML nach Simcron

Der Simcron-Modeller ist ein Produkt der Simcron GmbH und ein ereignisorientiertes Simulationswerkzeug, das speziell für den prozessbegleitenden Einsatz in der Fertigungsplanung konzipiert wurde. Modelle können mit einer graphischen 2-D-*Drag & Drop*-Oberfläche modelliert und simuliert werden. Die Anzahl der Bausteine wurde auf ein Minimum beschränkt. Zum Modellieren der Abläufe stehen neben den Bausteinen auch Skripte bereit, die mit Simskript bearbeitet werden können. Zum Austausch der Modelle dienen einfache Textdateien. Ein Szenario besteht aus den drei Textdateien szenarioX.mc zum Erzeugen der Objekte des Simulationsmodells, szenarioX.mp zur Parametrisierung der Objekte und szenarioX.ma zur graphischen Anordnung der Objekte. Die Transformation wurde für die Version Simcron Modeller 3.1 entwickelt.

4.4.1 Modellierungskonzept

Simcron-Modelle bestehen aus den drei Bausteingruppen `Ressource`, `Job` und `Technologie`. Das Element `Ressource` wird wiederum in `Maschinen` und `Warteschlangen` unterteilt. Die `Warteschlangen` können für die Darstellung des Abgangsprozesses, Ankunftsprozesses und der Warteschlange genutzt werden. `Maschinen` werden für die Abbildung von Ressourcen genutzt. Der `Job` stellt im Simcron Modeller das *Entity* dar. „Der Job ist das Gegenstück zu den Ressourcen-Bausteinen Warteschlange und Maschine. Während letztere die logistischen Kategorien Raum und Zeit anbieten, tritt der Job als Raum- und Zeit-Verbraucher auf. Jobs dienen in erster Linie dazu, Pro-

dukte – also den eigentlichen Gegenstand des Fertigungsprozesses – abzubilden“ (vgl. Simcron GmbH, 2003).

Im Element `Technologie` wird der Ablauf des *Entitys* und somit das Verhaltensmodell beschrieben. Die einzelnen Arbeitsschritte eines *Entities* innerhalb einer Technologie bezeichnen sich als `Arbeitsgänge`. „Die Technologie beschreibt eine Folge von Arbeitsgängen und stellt eine virtuelle Verknüpfung von Stationen (Maschinen- und Lagerbausteinen) her. Die einzelnen Arbeitsschritte werden ähnlich wie in einem Arbeitsplan aufgelistet. Die Ordnung in der Arbeitsgangliste entspricht der technologischen Reihenfolge im Fertigungsprozess“ (vgl. Simcron GmbH, 2003). Einem `Arbeitsgang` können `Maschinen, Warteschlangen` oder `Verzweigungen` zugeordnet werden. Verzweigungen sind mit den `Decision` und `MergeNodes` des SysML-Verhaltensmodells vergleichbar und ermöglichen die logische Auswahl von Folgeelementen. Der `Arbeitsgang` befindet sich nach der Einordnung des im Kapitel 3 vorgestellten SysML-Modellierungskonzeptes zwischen dem strukturellen und Verhaltens-Modell. Zum einen stellt er den Prozess dar, zum anderen aber auch die Oder-Verknüpfung des Verhaltensmodells. Ein weiterer semantischer Unterschied besteht darin, dass der Prozess `Arbeitsgang` nicht ohne die Assoziation einer `Maschine` modelliert werden kann. Darauf wird später detailliert eingegangen. Zudem gibt es das Element `Stochastik`, das erlaubt, Verteilungsfunktionen zu definieren, um sie dann mit Eigenschaften wie Störungen oder Prozesszeiten zu verknüpfen.

Die Semantik des Verhaltens bzw. der `Technologie` ist an die von Petri-Netzen angepasst, womit sie dem Verhaltensmodell dieser Arbeit entspricht. Die Ausführung eines `Arbeitsganges` wird erst dann begonnen, wenn alle vorausgegangenen `Arbeitsgänge` beendet worden sind. Abbildung 4-9 zeigt ein kleines Beispiel, in dem die beiden Jobs `job1` und `job2`, deren Ablauf in `tech1` und `tech2` festgelegt ist, ein Szenario durchlaufen. Die Jobs betreten das System durch die Ressource `Quelle`, werden im `Buffer` gelagert, auf dem Prozess `Maschine` bearbeitet und verlassen das System durch die `Senke`. `Maschine` stellt in diesem Szenario einen Prozess dar, der eine `Ressource` zum Ausführen benötigt.

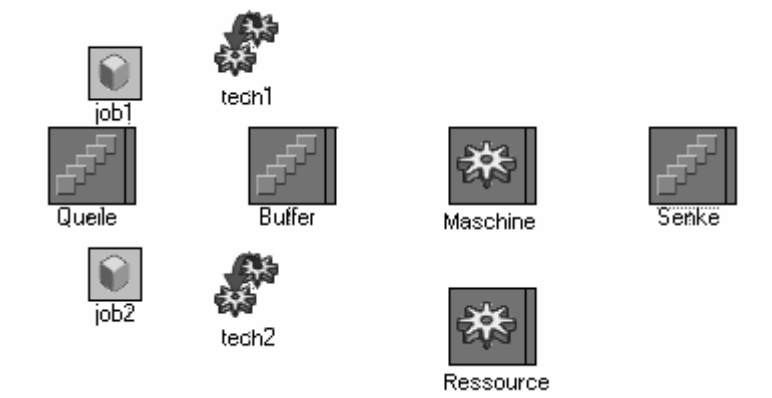


Abbildung 4-9: Beispiel Simcron

4.4.2 Transformation

In Tabelle 4-3 sind die Abbildungen der Elemente des statischen und Verhaltensmodells auf dem Simcron-Modeller zusammengefasst. Da sich die einzelnen Ressourcentypen des statischen Modells in ihrer Semantik kaum unterscheiden, wurden sie nicht gesondert aufgelistet. Die dynamischen Elemente sind im Modellierungskonzept dieser Arbeit oft in ihrer Semantik mit Elementen des statischen Modells verknüpft. Daher wurden bei der Übersicht der Transformationen der `FinalNode` (Abgangsprozess), der `InitialNode` (Ankunftsprozess), die `Action` (Prozess, Warteschlange) und die `ActivityPartition` (Zuordnung von Verhalten zur Struktur) nicht extra aufgeführt. Die genannten Verhaltenselemente werden durch die ihnen zugeordneten statischen Elementen implizit dargestellt. Beim Verhaltensmodell sind daher für die Transformation lediglich die `Edges` (Kanten), die `Merge` und `DescisionNodes` (Oder-Verknüpfungen) sowie die `Fork` und `JoinNodes` (Und-Verknüpfungen) zu betrachten.

Die Elemente *Entity*, Ankunftsprozess, Abgangsprozess und Warteschlange konnten fast ohne semantische Abweichungen transformiert werden. Der einzige Unterschied ist, dass der Ankunftsprozess, der Abgangsprozess und die Warteschlange vom gleichen Simcron Element `Warteschlange` abgebildet werden. Zudem war es möglich die meisten Attribute der Elemente ohne semantische Änderungen zu übernehemen. Auf die Umsetzung von *Batches* und Zeiteinheiten wurde bei der Transformation verzichtet. Bei den definierten Verteilungen der Enumeration Distribution des Ausgangsmodells war es möglich, alle ohne semantische Abweichungen nach Simcron zu transformieren.

Tabelle 4-3: Transformationsübersicht SimcronModeller

Modellierungskonzept	Simcron Modeller
Statische Elemente	
Ankunftsprozess	Warteschlange
Abgangsprozess	Warteschlange
Prozess	Technologie
Modus	Bausteinkombination
Warteschlange	Warteschlange
Ressource	Maschine
Relation	Technologie
Entity	Job
Dynamische Elemente	
Verknüpfung	Technologie
Und-Verknüpfung	Skript
Oder-Verknüpfung	Verzweigung

Die Relationen der Elemente werden, wie einleitend beschrieben, mit `Arbeitsgängen` dargestellt. Die Relationen zwischen den Prozessen und Ressourcen lassen sich in den `Arbeitsgängen` einer `Technologie` tabellarisch zuordnen. Das Verhalten des *Entitys* wird in Simcron mit Hilfe des Elements `Technologie` dargestellt. Die `DecisionNodes` bzw. `MergeNodes`, die optionale Wegeentscheidungen im SysML-Modellierungskonzept darstellen, konnten durch den Simcron `Verzweigungsbaustein` äquivalent transformiert werden.

Da das Kontrollmodell bei der Transformation nicht berücksichtigt wurde, können die Entscheidungen bei `Decision` und `MergeNodes` aktuell nur mit Wahrscheinlichkeiten nach Simcron transformiert werden. Entscheidungen bei konkurrierendem Zugriff auf Ressourcen regeln die Prioritäten des Elements Prozess, auch hier gibt es keine semantischen Probleme.

Wie ausgeführt, wird die Relation vom Prozess auf Ressourcen in den `Arbeitsgängen` abgebildet. Dabei verweisen die `Arbeitsgänge` auf `Maschinen`. Bei der Abbildung eines Prozesses, der keine Ressourcen nutzt, ist zu erwarten, dass ein `Arbeitsgang` ohne Verweis auf eine `Maschine` abgebildet wird. Dies wird von Simcron jedoch nicht gestattet und als ungültiges Modell erkannt. In der Transformation wird daher eine Warteschlange mit Bearbeitungszeit erstellt. Die Umsetzung enthält keine semantischen Einschränkungen, kann aber für den Nutzer irritierend sein.

Ressourcen lassen sich alle durch das Element `Maschine` umsetzen. Arbeiter, Transporter, Maschinen und Hilfsressourcen werden somit alle auf ein Element abgebildet. Keine Berücksichtigung fanden bei der Transformation spezielle Attribute der Ressourcengruppen, wie die Geschwindigkeit der Transporter oder die Qualifikation der Arbeiter.

Modi stellen verschiedene Konfigurationen von Prozessen dar (vgl. Kapitel 3.3). Dieser Sachverhalt kann auch abgebildet werden, indem die verschiedenen Modi als Prozesse dargestellt und durch eine Oder-Verknüpfung im Verhaltensmodell verbunden werden. So lassen sich die Modi auch in Simcron umsetzen. Die semantische Äquivalenz ist wie bei den genutzten Elementen Prozess und der Oder-Verknüpfung gegeben.

`Fork`- und `Join`-Verknüpfungen, die das logische „Und“ umsetzen und für die parallele Ausführung von Prozessen genutzt werden sollen, sind in Simcron nicht vorgesehen. Der `Fork` und `JoinNode` wurde durch die Nutzung zweier Warteschlangen (jeweils eine für `Fork` und eine für `Join`) und ein Simscript umgesetzt. Innerhalb des Skriptes wird der Job geteilt, synchronisiert und wieder zusammengeführt.

Die Kapselung von Diagrammen, die im SysML-Modellierungskonzept dieser Arbeit durch `CallbehaviorActions` umgesetzt wird (vgl. Kapitel 3.4), konnte nicht nach Simcron übertragen werden, da im Programm keine Möglichkeit für die Kapselung vorgesehen ist. Die Simcron Modellelemente sind im Gegensatz zu den SysML-Modellen mit Koordinaten im zweidimensionalen Raum belegt. Die Simcron Modellelemente lassen sich anhand ihrer Abfolge im SysML-Aktivitätsdiagramm anordnen. Die Ergebnisse sind detailliert in einer zugrunde liegenden Belegarbeit zusammengefasst, in dieser können auch Details zur Implementierung eingesehen werden (vgl. Zirakadze, 2010).

4.5 Modelltransformation von SysML nach Anylogic

AnyLogic von der Firma XJ Technologies ist eine Simulationssoftware, mit der diskrete und kontinuierliche Prozesse dargestellt werden können. Zusätzlich ist die Modellierung von Agenten möglich. AnyLogic ist sehr breit aufgestellt und entzieht sich der Spezialisierung auf einen bestimmten Teilbereich. Durch die Möglichkeiten, Agenten zu modellieren, können beispielsweise Mitarbeiter einer Fertigung individuell gestaltet werden. AnyLogic wurde mit Java entwickelt und orientiert sich stark am objektorientierten Modellentwurf, was die Entwicklung von größeren Projekten unterstützt. Die Transformation wurde für die Version AnyLogic 6.2.2 implementiert.

4.5.1 Modellierungskonzept

AnyLogic ist sehr stark an die Java Entwicklungsplattform Eclipse angelehnt. Ein erfahrener Java-Programmierer findet sich daher schnell in der AnyLogic-Oberfläche zurecht. Es stehen für verschie-

dene Probleme einzelne Bibliotheken mit untergeordneten Arbeitsobjekten zur Auswahl. Die Auswahl der Arbeitselemente ist von allen untersuchten Programmen am umfangreichsten, was sicher nicht zuletzt an der starken Generalisierung liegt.

In der Anylogic-Standard-Bibliothek `Model` stehen Standard-Elemente wie `Variablen, Events, Funktionen, States, Branches, Ports` und `Connectors` zur Verfügung. In der Bibliothek `Action` wird die Modellierung von Verzweigungen und Schleifen gestattet (beispielsweise `Decision, While, Do While, For`). Mit den Elementen der ersten beiden Bibliotheken ermöglicht AnyLogic eine sehr freie Modellierung grundlegender Sachverhalte, die durch ihre feine Granularität aber auch einen erhöhten Modellierungsaufwand mit sich bringt.

In der `Enterprise Library` setzt AnyLogic in der Granularität der Modellierung eine Stufe höher an als bei der Standard-Bibliothek und stellt grob granularere Elemente für die Modellierung bereit. Da diese Granularitätsstufe der des Modellierungskonzeptes dieser Arbeit am nächsten ist, wurde die `Enterprise Library` für die Transformation nach Anylogic genutzt. So kann der Ankunftsprozess als `Source`, der Abgangsprozess als `Sink`, der Prozess als `Delay`, die Warteschlange als `Queue` und der Ressourcen- Pool als `Resource Pool` modelliert werden. Zusätzlich gibt es spezialisierte Elemente, die den Modellierungsaufwand verringern können, deren Abbildung aber auch mit den gerade beschriebenen Elementen möglich ist (`Batch, Unbatch, Dropoff, Pickup, Conveyor`). Für die Modellierung des Verhaltens gibt es Verbindungen der Elemente (`Connectors`) und Verzweigungselemente (`Select Output, Split, Combine`). Die Semantik der Elemente des Verhaltens ist der von Petri-Netzen angepasst und somit dem Verhaltensmodell dieser Dissertation ähnlich. Das Verhalten, die Struktur sowie die Steuerung sind in Anylogic ineinander vermischt.

Die einzelnen Elemente haben eine Vielzahl von Attributen, zudem ist es möglich, den *Entities* neue Attribute zuzuweisen. Jedes Element kann bei Eingang, der Bearbeitung oder bei Ausgang eines *Entities* eingefügten Java Code ausführen. Auch für das *Dispatching* können eigene Funktionen geschrieben werden. Zudem sind die gebräuchlichen Verteilungsfunktionen wiederzufinden.

Abbildung 4-10 zeigt ein Beispiel für ein Anylogic-Modell. Ein Werkstück betritt das System durch das Element `Source`. Dann wird es durch das Element `SelectOutput` an die obere oder untere Fertigungsstrecke verwiesen. In jeder der beiden Fertigungsstrecken wird das *Entity* drei Mal in einer Warteschlange gelagert und anschließend von einem Prozess bearbeitet. Die Prozesse greifen bei ihren Bearbeitungsvorgängen auf Ressourcen zu. So benötigen beispielsweise die vier Prozesse `wait_for_transporter` die Ressource `ressourcetransporter, die als Ressourcenpool` modelliert ist. Am Ende einer Fertigungsstrecke wird über `selectOutput1` oder `selectOutput2` entschieden, ob das *Entity* zum Ausgang des Szenarios `sink` oder zur gegenüberliegenden Fertigungsstrecke weiterläuft.

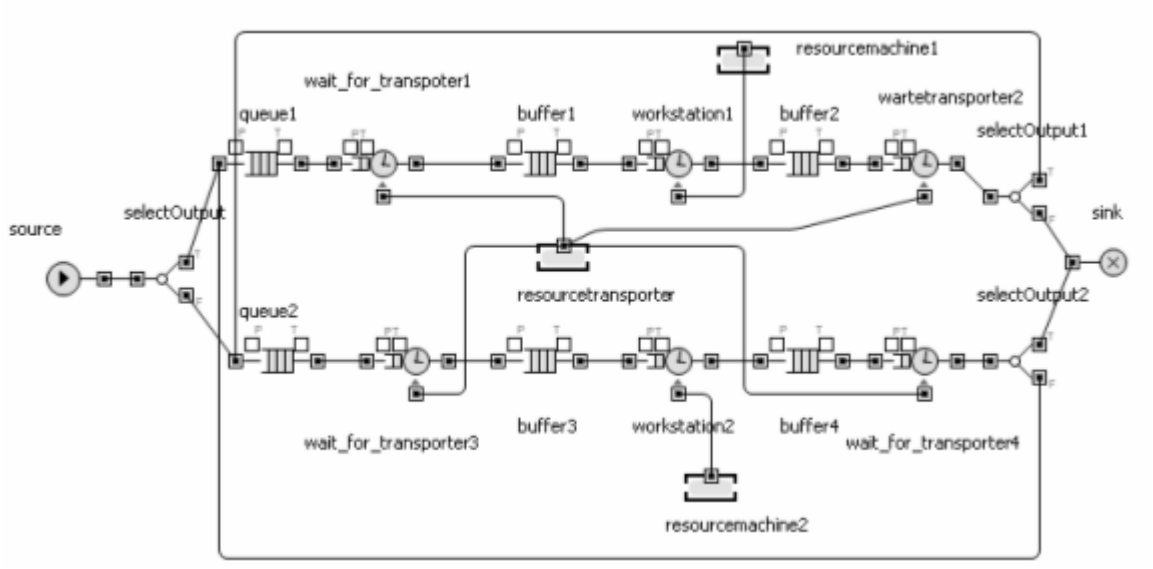


Abbildung 4-10: Beispielszenario AnyLogic

4.5.2 Transformation

Tabelle 4-4 gibt einen Überblick über die transformierten Elemente. Wie bei der Betrachtung des Simcron-Modellers wurden die verschiedenen Ressourcentypen aus Redundanzgründen im Typ Ressource zusammengefasst und nur die Verknüpfungen und booleschen Operationen für die Transformation des Verhaltensmodells betrachtet.

In Anylogic können der Ankunftsprozess, der Abgangsprozess, die Warteschlange und das *Entity* ohne semantische Änderungen transformiert werden. Auch die Kanten und Und-Verknüpfungen des Verhaltensmodells können ohne semantische Einschränkungen transformiert werden. Die Modelle des SysML-Ausgangsmodells unterscheiden sich in der Darstellung des Zielmodells in Anylogic, da in Anylogic die Abbildung des Verhaltens und der Struktur gemeinsam erfolgt. Durch die Vermengung entstehen jedoch keine semantischen Verfälschungen. Es war auch möglich, die Attribute der Elemente größtenteils ohne semantische Änderungen zu transformieren. Lediglich auf das Umsetzen der Batchsize und der Attribute der Assoziation im Ausgangsmodell wurde verzichtet.

Wie im Modellierungskonzept dieser Arbeit können Ressourcen durch `Ressourcenpools` modelliert werden. Prozesse, die keine Ressourcen benötigen, bildet Anylogic als `Delays` ab, Prozesse, die Ressourcen benötigen, als `Services`. Doch ist es nicht möglich, mit den in Anylogic angebotenen Standardelementen Prozesse zu modellieren, die verschiedene Ressourcen benötigen. Um dies durchzusetzen, sind komplexe Modellierungsabläufe notwendig, die Yang in einer Masterarbeit erörtert (Yang, 2010, S. 20 ff.). Die erarbeiteten Muster könnten dann als Entwurfsmuster für die Modelltransformation genutzt werden, in der aktuellen Version des Übersetzers wurde dies jedoch nicht berücksichtigt.

Tabelle 4-4: Transformationsübersicht Anylogic

Modellierungskonzept	Anylogic
Statische Elemente	
Ankunftsprozess	Source
Abgangsprozess	Sink
Prozess	Delay/Service
Modus	Bausteinkombination
Warteschlange	Queue
Ressource	Resourcepool
Relation	Connector
Entity	Entity
Dynamische Elemente	
Verknüpfung	Connector
Und-Verknüpfung	Select Output
Oder-Verknüpfung	Split/Combine

Für die Modellierung von Oder-Verknüpfungen kann das Anylogic-Element `Select Output` genutzt werden. Jedoch stellt das Element `Select Output` nur zwei Ausgangsports bereit. Zwar ist es möglich, mehrere `Select Output's` aneinanderzufügen, doch ist es nicht mehr möglich die Ausgangsports bei der automatischen Modellerzeugung eindeutig deren Nachfolgern zuzuweisen. Um dieses Problem zu umgehen, müssen zusammengesetzte `Select-Output`-Elemente als Entwurfsmuster vorgefertigt werden, für jede Anzahl an Ausgängen ein gesondertes Muster. Mit den Entwurfsmustern ist es möglich, `Select-Output`-Elemente mit einer beliebigen Anzahl an Ausgängen zur Transformation der Oder-Verknüpfungen ohne semantische Änderungen umzusetzen.

Ein weiteres semantisches Problem tritt auf, wenn ein *Entity* in Anylogic an ein Element geleitet wird das voll ausgelastet ist. Beispiele wären ein arbeitendes `Delay` oder eine voll belegte `Queue`. Wird ein Durchflussobjekt an ein solches Element geleitet, gibt Anylogic eine Fehlermeldung und bricht die Simulation ab. Die Behandlung eines solchen Problems ist im Modellierungskonzept dieser Arbeit nicht definiert. Eine mögliche Lösung wäre, das vorhergehende Element im Kontext eines *Pull*-Modells auf die Freigabemeldungen seines Nachfolgers warten zu lassen. Eine andere Lösung bestünde darin, das gesendete *Entity* bei der Umsetzung eines *Push*-Systems in einer imaginären Warteschlange mit unbegrenztem Speicher einzulagern. So könnte zwischen den einzelnen Elementen eine `Queue` modelliert werden, was jedoch zu semantischen Änderungen führt.

Wie Simcron stellt auch Anylogic keine Modi bereit. Deren Umsetzung kann ähnlich wie bei Simcron durch das Kombinieren mehrerer `Select Outputs` mit `Services` bzw. `Delays` durchgeführt werden. Die Hierarchisierung der Modelle, die im Konzept dieser Arbeit durch `CallBehaviorActions` umgesetzt wird, kann in Anylogic auch durch einfache Kapselung der Diagramme semantisch äquivalent umgesetzt werden. Für die graphische 2-D Umsetzung erfolgt die Anordnung der Elemente einfach baumartig von der Senke an.

4.6 Modelltransformation von SysML nach Flexsim

Das Programm Flexsim wird seit 1993 von der amerikanischen Firma *Flexsim Software Products*, Inc., entwickelt und wird zum Optimieren, Analysieren und Planen von diskreten Prozessen eingesetzt. Mit Flexsim lassen sich Modelle auf einer 2-D-*Drag-and-Drop*-Oberfläche entwickeln und in 2-D sowie 3-D simulieren. Zudem können in Flexsim kontinuierliche und diskrete Prozesse simuliert werden. Das Programm wurde für die Modellierung verschiedenster Domänen, aber vor allem für die Domänen Logistik, Produktion sowie *Healthcare* entwickelt. Die Transformation wurde für die Version Flexsim 4.4 umgesetzt.

4.6.1 Modellierungskonzept

Flexsim stellt eine breite Menge von Elementen, darunter die grundlegenden Elemente Ankunftsprozess (`Quelle`), Abgangsprozess (`Senke`), Warteschlange (`Puffer`), Prozess (`Maschine`), sowie die beweglichen Ressourcen (`Personal`) und (`Transporter`). Auf die Modellierung von Ressourcenpools wurde verzichtet, wodurch die Modelle schnell groß werden können. Zusätzlich werden verschiedene Hilfselemente, wie beispielsweise `Roboter`, `Fließbänder`, `Separator` oder `Multiprozessor` zur Modellierung angeboten. Weiterhin gibt es eine spezielle Bibliothek mit zehn Elementen zur Modellierung flüssiger Stoffe. Für die beweglichen Ressourcen errechnen sich Streckenzeiten durch Zielentfernung und Geschwindigkeit. Zur Bewältigung damit verbundener Umstände gibt es `Networknodes`, durch die Strecken mit Wegzeiten belegt werden können.

Jedes Element besitzt verschiedene Ereignispunkte, die beim Eintreten eines bestimmten Ereignisses eine geschriebene Funktion ausführen. So besitzt die Maschine die Ereignispunkte `On-Reset, On-Message, On-Entry, On-Exit` und `On-Finish`. Ein `Operator` hingegen besitzt die für ihn wichtigen Ereignispunkte `On-Load` und `On-Unload`. So passt Flexsim die Ereignispunkte, im Gegensatz zu allen anderen Simulatoren, individuell den Elementen an. Funktionen können in Flexscript, der Simulationssprache von Flexsim, geschrieben werden, das sich an C++ orientiert. Optional gibt es die Möglichkeit, Funktionen auch in C++ zu schreiben. Diese müssen dann aber vor der Ausführung kompiliert werden.

Abbildung 4-11 zeigt ein in Flexsim abgebildetes Beispielszenario. Links schleust der Ankunftsprozess `ap` die Werkstücke in das Modell ein. Der `ForkNode10` leitet die *Entities* zum Prozess `sp1` oder `sp2` weiter. Der Prozess `sp2` benötigt eine Maschine `m1` um starten zu können. Abschließend werden die Werkstücke über `JoinNode10` zum Abgangsprozess `dp` geleitet.

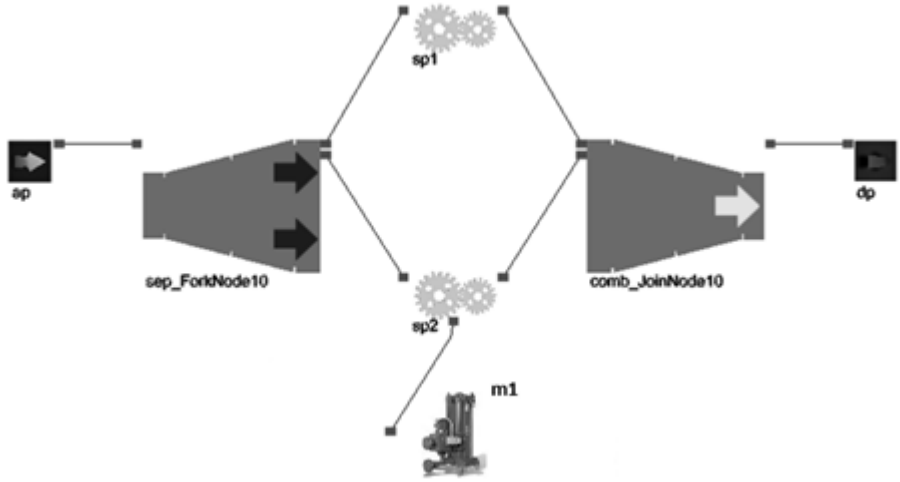


Abbildung 4-11: Beispielszenario Flexsim

4.6.2 Transformation

In der Tabelle 4-5 wird die Transformationszuordnung für die Übersetzung von SysML nach Flexsim gezeigt. Wie in allen Zusammenfassungen der Transformationen wurden beim Verhaltensmodell nur die Verknüpfungen und Booleschen Operationen betrachtet. Auch die verschiedenen Ressourcentypen wurden aus Redundanzgründen zusammengefasst. Im Gegensatz zu den anderen Simulationswerkzeugen bietet Flexsim jedoch durch die einleitend eingeführten `Networknodes` ein Raumkonzept. Zwischen zwei `Networknodes` die jeweils mit einen Element verbunden sind, ist es möglich, Entfernungen anzugeben. Die Ressource Transporter kann dadurch als bewegliche Ressource mit den Attributen Geschwindigkeit und Kapazität modelliert werden. Um das Modellierungskonzept der Arbeit zu nutzen, kann zwischen zwei Maschinen ein Transportprozess angegeben werden, dessen Zeitdauer bei einer Durchschnittsgeschwindigkeit von beispielsweise 10 m/s festgelegt wird. Nun kann der Transformator die Entfernung zwischen den Maschinen berechnen, um `Networknodes` zwischen den Maschinen zu erstellen. Die graphische Darstellung eines Szenarios in Flexsim wird trotzdem einfach aus den Zusammenhängen der Prozesse des Aktivitätsdiagramms als Baum dargestellt.

Tabelle 4-5: Transformationsübersicht Flexsim

Modellierungskonzept	Flexsim
Statische Elemente	
Ankunftsprozess	Quelle
Abgangsprozess	Senke
Prozess	Maschine
Modus	Bausteinkombination
Warteschlange	Puffer
Ressource	Personal, Transporter
Relation	Connector
Entity	Entity
Dynamische Elemente	
Verknüpfung	Connector
Und-Verknüpfung	Separator/Combiner
Oder-Verknüpfung	Split/Combine

Ein Großteil der Elemente und ihrer Eigenschaften konnte ohne semantische Schwierigkeiten nach Flexsim übertragen werden. Die Eigenschaften *batchSize, priority, modePriority* und *capacity* der jeweiligen Elemente sind schwer umzusetzen. Obwohl die Umsetzung der Attribute möglich ist, wurde sie im Übersetzer noch nicht berücksichtigt. Es ist nicht möglich, die Eigenschaften der *Extended-Association* mit den angebotenen Flexsim Elementen und deren Eigenschaften auszudrücken.

Die *Fork*- und *Join*-Verknüpfungen lassen sich durch die Flexsimelemente `Separator` und `Combiner` darstellen. Im Ausgangsmodell kann das *Fork* Element mit beliebig vielen Ausgängen und ohne Kapazität dargestellt werden. In Flexsim wird für jeweils zwei Ausgänge ein gesondertes Element mit einer Kapazität von eins benötigt. Die Oder-Verknüpfungen hingegen ließen sich auch mit einer größeren Zahl an Ausgängen beispielsweise im Gegensatz zu Anylogic ohne Probleme umsetzen.

Wie die anderen Simulationswerkzeuge, die für die Übersetzer entwickelt wurden, bietet auch Flexsim keine Modi an. Auch in Flexsim können die Modi transformiert werden, indem die gleiche Anzahl von Prozessen wie Modi abgebildet und mit Oder-Verknüpfungen verbunden wird. Flexim arbeitet nicht mit Ressourcenpools, sondern mit einzelnen Ressourcen. Jedoch ist es möglich, diesen einen `Dispatcher` zuzuweisen, der die Ressourcen verwaltet und wie ein `Ressourcepool` agiert.

Flexsim bietet zudem auch ein Konzept für Hierarchisierungen, auf das die `Callbehavior` Aufrufe des Zielmodells abgebildet werden konnten. Die 2-D Darstellung wird auch in Flexsim baumartig

aufgebaut. Der Übersetzer für Flexsim ist aus einer zugrunde liegenden Belegarbeit entstanden (vgl. Mudo, 2011).

4.6.3 Rückwärtstransformation von Flexsim nach SysML

Neben der Transformation von SysML nach Flexsim wurde eine Übersetzung von Flexsim nach SysML entwickelt. Das Übersetzer-*Plugin* nimmt die Informationen vom Flexsim-Export und bereitet sie für das interne Modell auf (vgl. Kapitel 4.3.3). Anschließend kann ein Programm die Informationen vom internen Modell nach TOPCASED *Engineer* übersetzen. Die Modelle werden in den Aktivitätsdiagrammen des Zielmodells dann baumartig in Form von Aktivitätsdiagrammen dargestellt.

Die Übersetzung ist mit der Vorwärtstransformation äquivalent. Es können alle Elemente von Flexsim übersetzt werden, auf die in der Vorwärtstransformation abgebildet wird (vgl. Tabelle 4-5). Die semantischen Probleme der Transformation von SysML nach Flexsim bleiben daher bei der Transformation von Flexsim nach SysML bestehen. Die Arbeit an einem Übersetzer, der die vollständigen Modellierungsmöglichkeiten von Flexsim auf das Modellierungskonzept dieser Arbeit überträgt, wäre um ein Vielfaches größer, da die Modellierungsmöglichkeiten von Flexsim umfangreicher sind. Ein Teil der Elemente von Flexsim sind für andere Domänen spezifiziert. Zudem würde der Aufwand der Übersetzung von Modellteilen, die mit der Flexsim-Modellierungssprache Flexscript entwickelt sind, den Nutzen übersteigen.

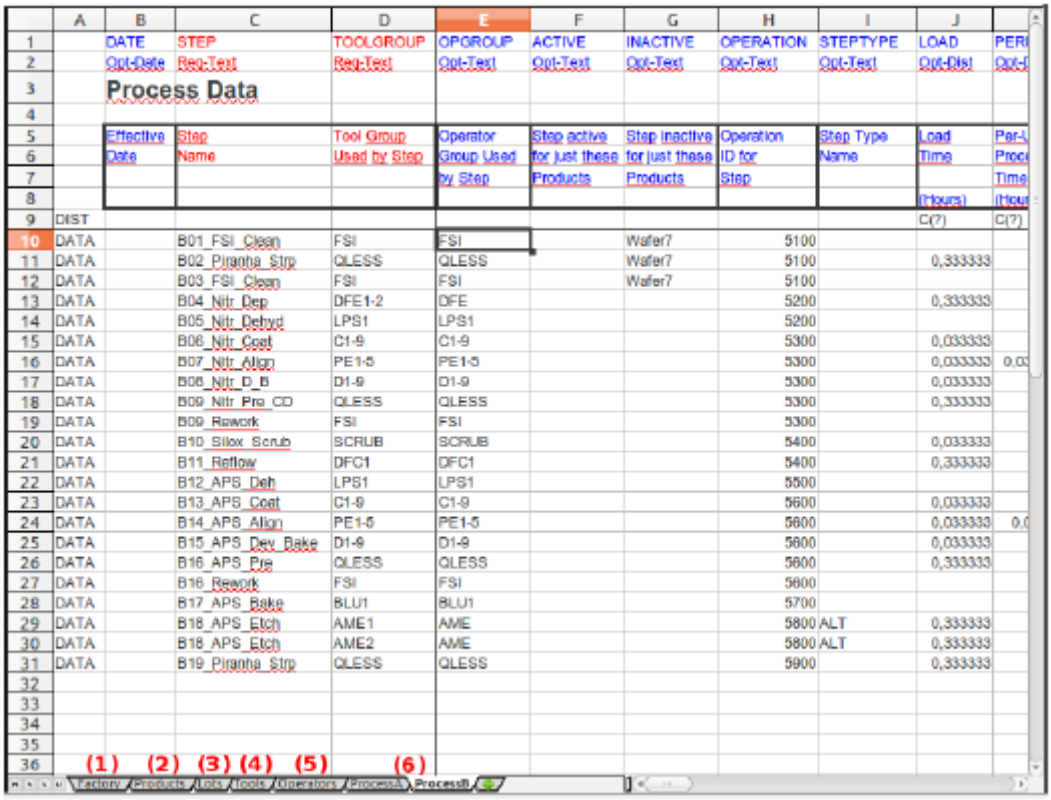
4.7 Modelltransformation von SysML zu Factory Explorer

Das Simulationswerkzeug Factory Explorer wird von Wright Williams & Kelly, Inc., angeboten und ist „ein integriertes Kapazitäten-, Kosten- und Durchlaufzeiten Analysetool für Fabriken“ (vgl. Wright Williams & Kelly, 2014). Mit dem Werkzeug können Fabriken als Excel-Tabellen (Workbooks) modelliert, simuliert und analysiert werden. Die Darstellung von Excel-Tabellen unterscheidet sich von der Darstellung der anderen genutzten 2-D *Drag-and-Drop*-Tools. Um den Factory Explorer nutzen zu können, wird eine Microsoft-Excel-Lizenz benötigt. Das Werkzeug ist speziell für die Modellierung der Ablaufprozesse von Fabriken entwickelt. Neben Durchlaufzeiten und Auslastungen können auch Kosten modelliert, simuliert und überprüft werden. Die Transformation wurde für die Version Factory Explorer 2.8 entwickelt.

4.7.1 Modellierungskonzept

Ein Factory Explorer Modell in Form eines *Workbooks* besteht aus einer Menge von *Worksheets*, in denen jeweils ein bestimmter Teil eines Modells beschrieben wird. Die tabellarische Darstellung der Modelle unterscheidet sich auf den ersten Blick von der üblichen 2-D *Drag-and-Drop* Darstellung,

kann aber semantisch genauso Elemente, Relation und deren Attribute ausdrücken. In Abbildung 4-12 ist ein Beispielmodell einer Halbleiterfabrik als *Worksheet* abgebildet.



	A	B	C	D	E	F	G	H	I	J	
1		DATE	STEP	TOOLGROUP	OPGROUP	ACTIVE	INACTIVE	OPERATION	STEPTYPE	LOAD	PERI
2		Opt-Date	Req-Text	Req-Text	Opt-Text	Opt-Text	Opt-Text	Opt-Text	Opt-Text	Opt-Dist	Opt-D
3		Process Data									
4											
5		Effective	Step	Tool Group	Operator	Step active	Step inactive	Operation	Step Type	Load	Per-L
6		Date	Name	Used by Step	Group Used	for just these	for just these	ID for	Name	Time	Proc
7					by Step	Products	Products	Step			Time
8										(Hours)	(Hour
9	DIST									C(7)	C(7)
10	DATA		B01_FSI_Clean	FSI	FSI		Wafer7	5100			
11	DATA		B02_Piranha_Strp	QLESS	QLESS		Wafer7	5100		0,333333	
12	DATA		B03_FSI_Clean	FSI	FSI		Wafer7	5100			
13	DATA		B04_Nitr_Dep	DFE1-2	DFE			5200		0,333333	
14	DATA		B05_Nitr_Dehyd	LPS1	LPS1			5200			
15	DATA		B06_Nitr_Coat	C1-9	C1-9			5300		0,033333	
16	DATA		B07_Nitr_Align	PE1-5	PE1-5			5300		0,033333	0,03
17	DATA		B08_Nitr_D_B	D1-9	D1-9			5300		0,033333	
18	DATA		B09_Nitr_Pre_CD	QLESS	QLESS			5300		0,333333	
19	DATA		B09_Rework	FSI	FSI			5300			
20	DATA		B10_Silox_Scrub	SCRUB	SCRUB			5400		0,033333	
21	DATA		B11_Reflow	DFC1	DFC1			5400		0,333333	
22	DATA		B12_APS_Deh	LPS1	LPS1			5500			
23	DATA		B13_APS_Coat	C1-9	C1-9			5600		0,033333	
24	DATA		B14_APS_Align	PE1-5	PE1-5			5600		0,033333	0,0
25	DATA		B15_APS_Dev_Bake	D1-9	D1-9			5600		0,033333	
26	DATA		B16_APS_Pre	QLESS	QLESS			5600		0,333333	
27	DATA		B16_Rework	FSI	FSI			5600			
28	DATA		B17_APS_Bake	BLU1	BLU1			5700			
29	DATA		B18_APS_Etch	AME1	AME			5800	ALT	0,333333	
30	DATA		B18_APS_Etch	AME2	AME			5800	ALT	0,333333	
31	DATA		B19_Piranha_Strp	QLESS	QLESS			5900		0,333333	
32											
33											
34											
35											
36		(1)	(2)	(3) (4) (5)	(6)						

Abbildung 4-12: Worksheet eines Factory-Explorer-Modells

Wie im Modellierungskonzept dieser Arbeit wird das Modell durch die *Worksheets* in verschiedene Teilmodelle getrennt, die jedoch eine andere Abgrenzung und Sichtweise nutzen (vgl. Abbildung 4-12). Ein Modell besteht aus den folgenden *Worksheets*:

1. *Factory*: Beschreibt Detailwissen über eine Betriebsstätte (beispielsweise Fixkosten und wiederkehrende Kosten).
2. *Products*: Beschreibt Detailwissen über die Produkte (beispielsweise Ankunftsraten und Größe der Lose).
3. *Lots*: Beschreibt die individuellen Losgrößen.
4. *Tools*: Beschreibt die verschiedenen Werkzeuggruppen (beispielsweise Stückzahl und Unterbrechungen wie Ausfälle und Wartungen).
5. *Operators*: Beschreibt die Ressourcen (beispielsweise Anzahl und Arbeitsunterbrechungen).
6. *Process*: Beschreibt den Prozessfluss für ein Produkt durch einen Arbeitsplan mit allen Prozessschritten bzw. Operationen.

Jedes Modell muss die beschriebenen *Worksheets* enthalten, zudem muss für jedes definierte Produkt ein `Process Worksheet` definiert werden, das seinen Prozessfluss beschreibt.

Da der Factory Explorer speziell für Prozesse in der Halbleiterindustrie entwickelt wurde, ist sein Modellierungskonzept der Domäne stark angepasst. So stehen beispielsweise `Operator` und `Tools` als Ressourcen zur Verfügung und die Durchlaufobjekte werden in `Units, Lots` und `Batches` unterteilt. Es werden auch spezielle Prüfmuster aus der Halbleiterindustrie übernommen und als Entwurfsmuster angeboten. So ist es möglich mit den Entwurfsmustern `Scrap` und `Rework`, Durchlaufobjekte mit festgelegten Verfahren, die sich über mehrere Prozesse erstrecken, auf Ausschuss oder Wiederbearbeitung zu überprüfen. Hingegen können einfache logische Und-Verknüpfungen mit dem Factory Explorer nicht abgebildet werden. Da das Werkzeug speziell für die Domäne der Halbleiterfertigung entwickelt wurde, hat es ein sehr spezifisches Modellierungskonzept.

Ein Szenario enthält wie beschrieben verschiedene *Entities* (`Products`), deren Verhalten in einem besonderen *Worksheet* beschrieben wird. In diesem werden die Prozesse (`Steps`), die ein `Product` durchläuft, untereinander aufgeführt. Ein `Product` wandert immer von einem `Step` zu dem `Step`, der in der nachfolgenden Zeile aufgeführt ist. Somit beschreibt der Sprung von einer Zeile in die nächste die `Edges` der SysML-Aktivitätsdiagramme. Ist es notwendig `DecisionNodes` abzubilden, kann dies durch `Gotos` erfolgen, die bedingten Sprunganweisungen ähneln. Die Ressourcen lassen sich in den *Worksheets* `Tools` und `Operators` zeilenbasiert umsetzen. Während ein Element durch eine Zeile beschrieben wird, werden die Attribute den Elementen in ihren Spalten zugeordnet. Die Zuordnungen der Ressourcen zu den Prozessen, erfolgt auch in den Spalten der Prozesse.

4.7.2 Transformation

Tabelle 4-6 zeigt die Transformation der Elemente von SysML in den Factory Explorer. Wie bei den Ausführungen der anderen Transformationen, wurden die verschiedenen Ressourcentypen aus Redundanzgründen mit dem Typ Ressource zusammengefasst und nur die Verknüpfungen und booleschen Operationen für die Transformation des Verhaltensmodells betrachtet.

Das *Entity* wie auch der Ankunftsprozess werden im Factory Explorer beide durch das Element `Product` dargestellt. Die Werte `arrivalIntervalDefinition, arrivalIntervalDistribution` und `arrivalIntervalUnit` des `ArrivalProcess` werden beim `Product` im Wert `releaseInterval` abgebildet. Ist die Eingangswarteschlange beim `ArrivalProcess` durch eine `capacity` beschränkt, wird ein `Step` erzeugt, der den Sachverhalt darstellen kann. Der `DepartureProcess` wird im Factory Explorer nicht transformiert, da das *Entit*y das System nach dem letzten Prozessschritt automatisch verlässt.

Der Factory Explorer modelliert Fehler im Element `Interrupts`, das mit dem jeweiligen betroffenen Element verbunden werden kann. Sowohl `Operator` als auch `Tools` können `Interrupts` besitzen. Prozesse oder Warteschlangen können im Factory Explorer jedoch keinem Fehler unterliegen. Es ist möglich, die verschiedenen definierten Verteilungen alle im Factory Explorer abzubilden und beispielsweise auch für das Modellieren von Fehlern zu nutzen.

Im Factory Explorer besitzt jedes *Tool* standardmäßig einen unendlichen Eingangs- und Ausgangspuffer. Diese Puffer müssen zum Abgleich des Ausgangsmodells begrenzt werden. Jedoch liegt der niedrigste zugelassene Wert bei eins, daher haben alle Prozesse, die eine Maschine nutzen, einen Eingangspuffer und Ausgangspuffer der Kapazität eins, was semantisch zu Unterschieden führt. Um Warteschlangen abzubilden, werden ein `Tool` und ein `Step` benötigt. Der `Step` übernimmt den Namen der Warteschlange. Da ein `Tool` genutzt wird, beträgt die Mindestkapazität der Warteschlange durch den Eingangs- und Ausgangspuffer zwei.

Tabelle 4-6: Transformationsübersicht Factory Explorer

Modellierungskonzept	Factoryexplorer
Statische Elemente	
Ankunftsprozess	Product
Abgangsprozess	-
Prozess	Step
Modus	Step
Warteschlange	Kombination aus Step und Tool
Ressource	Operator, Tool
Relation	Connector
Entity	Product
Dynamische Elemente	
Verknüpfung	Nachfolgender Tabelleneintrag
Und-Verknüpfung	-
Oder-Verknüpfung	Goto

Die Ressource `Worker` wird auf das Factory Explorer Element `Operator` abgebildet. Alle anderen Ressourcentypen des Ausgangsmodells lassen sich auf das Element `Tool` abbilden. Für das Attribut `qualification` des `Workers` gibt es keine Übersetzungsmöglichkeit in Factory Explorer, auch die Attribute des Transporters können nicht übersetzt werden.

Das Element `SingleProcess` kann ohne Einschränkungen auf das Zielelement `Step` abgebildet werden. Wie ausgeführt, ist lediglich das Umsetzen von prozessbezogenen Fehlern nicht möglich. Auch Modi können ohne Probleme als Menge von Prozessen, die mit Oder-Verknüpfungen verbunden sind, dargestellt werden. Im Factory Explorer ist es notwendig, jedem `Step` genau ein *Tool* zuzuweisen. Daher ist es notwendig, Prozessen die keine Ressourcen benötigen, ein *Dummy Tool* zuzuordnen. Dadurch entstehen wieder semantische Unterschiede durch den nötigen Eingangs- und Ausgangspuffer des `Tools`. Weiterhin kann jedem `Step` nur ein *Tool* sowie ein `Operator` zugeordnet werden. Daher ist es notwendig, Prozesse die mehr Ressourcen benötigen, mit mehreren `Steps` abzubilden. Jedoch ist es nicht möglich, ein `Tool` zu reservieren und im nächsten Schritt ein gleichnamiges `Tool` zu reservieren, das gleiche gilt für `Operator`. Daher kann das Reservieren einer Anzahl von mehreren Ressourcen für einen Prozess nicht in Factory Explorer abgebildet werden.

`Decision`- und `MergeNodes` können durch die `Goto`-Spalte, die Sprungmarken enthält, abgebildet werden. Für `Fork`- und `JoinNodes` gibt es jedoch keine mögliche Abbildung im Factory Explorer, wodurch die Semantik erheblich eingeschränkt ist. Für die Darstellung werden die Elemente den *Worksheets* zugeordnet und die Prozesse sequenziell nach ihrer Anordnung im Aktivitätsdiagramm aufgereiht (vgl. Scharfe, 2011).

5 Verifikation und Validierung

Modelltransformationen liegen aktuell im Fokus der Informatikforschung, wobei insbesondere seit dem starken Anstieg des Anteils an modellgetriebener Softwareentwicklung Modelltransformationen an Bedeutung gewonnen haben. Neben der Entwicklung von Transformationsansätzen konzentriert sich die Forschung vertieft auf deren Verifikation und Validierung. Dementsprechend erfolgt eine Überprüfung der Modelltransformationen hinsichtlich definierter Kriterien auf Korrektheit. In diesem Kapitel soll die Verifikation und Validierung des vorgestellten Transformationsansatzes erarbeitet werden. Das ist für das Gesamtvorhaben wichtig, da die transformierten Modelle ansonsten in praktischen Arbeiten nicht genutzt werden könnten. Die Wahrscheinlichkeit, dass sie sich vom Ausgangsmodell unterscheiden, wäre zu hoch.

Um ein Konzept für die Verifikation und Validierung zu erarbeiten, werden zuerst einige Definitionen hergeleitet. Anschließend erfolgt die Erörterung von Kriterien, die bestimmen, ob ein Modell korrekt transformiert wurde. Nach einem Überblicke über Methoden zur Verifikation und Validierung ist es möglich, diese auf Anwendbarkeit für die erarbeiteten Zielkriterien zu überprüfen.

5.1 Grundlagen

Verifikation und Validierung werden in der Literatur oft als Synonym verwendet, obwohl beide Begriffe grundsätzlich verschiedene Ziele verfolgen (Schatten et al., 2010, S. 5). Während bei der Modellverifikation ein Modell auf seine Korrektheit geprüft wird, untersucht die Validierung die Nützlichkeit eines Modells (vgl. Mens, 2010), letzteres gibt darüber Auskunft, inwiefern Ergebnisse eines Modells mit dem realen Prozess korrelieren. Die Verifikation beantwortet die Frage: „Wurde das Modell korrekt entwickelt?“. Dagegen widmet sich die Modellvalidierung der Frage: „Wurde das korrekte Modell entwickelt?“ (vgl. Banks, 1998).

In Bezug auf die Modelltransformation finden sich verschiedenste Begriffsdefinitionen. Weit verbreitet ist die Sicht, die Validierung als Oberbegriff zu betrachten, die durch Testen und Verifikation erreicht werden kann (vgl. Mens, 2010; vgl. Küster/Abdelrazik, 2006). Dabei erfolgt die Überprüfung, ob aus der Transformation korrekte Modelle resultieren, die wichtige Eigenschaften wie beispielsweise semantische und syntaktische Korrektheit erhalten. Oft wird Testen auch als Synonym für nichtformale Verifikation genutzt (vgl. Bresser 2004). Die Abgrenzung zwischen Verifizieren und Testen ist in der Literatur nicht immer eindeutig.

In Cabot et al. (2010) wird die Verifikation und Validierung von Modellen analog zur Verifikation und Validierung von Modelltransformation gesehen. Validierung betrachtet hierbei die Frage: „Ist dies die korrekte Transformation?“ und Verifikation „Ist die Transformation korrekt?“. Bei der Validierung werden dabei Ergebnisse von Ausführungen der Transformationen auf verschiedene Testszenarien mit erwarteten Ergebnissen verglichen. Bei der Verifikation wird unter anderem das Finden von Fehlern bei der Transformation betrachtet. In dieser Arbeit soll Verifikation und Validierung wie folgt definiert werden:

Validierung von Modelltransformationen: „Die Validierung von Modelltransformationen ist ein Prozess, der überprüft, ob die Spezifikation der Transformation den Anforderungen an die Transformation genügt.“ (Scharfe, 2011, S. 57)

Verifikation von Modelltransformationen: „Die Verifikation von Modelltransformationen ist ein Prozess, welcher überprüft, ob die Umsetzung der Transformation ihrer Spezifikation genügt.“ (Scharfe, 2011, S. 57)

Für die im Kontext dieser Arbeit zu entwickelnden Modelltransformation, sind beispielsweise folgende Anforderungen definiert: Die Transformation soll

- von einem SysML-Modell auf ein Modell für ein spezifisches Simulationswerkzeug abbilden,
- semantisch, funktional und syntaktisch korrekt sein.

Die Regeln und Algorithmen zur Übersetzung der Elemente des SysML-Metamodells sind die Spezifikationen der Transformation. Dementsprechend wird bei der Validierung geprüft, ob die Algorithmen und Regeln den Anforderungen entsprechen. Die Verifikation hingegen, überprüft, ob die Implementierungen gemäß den aufgestellten Algorithmen und Regeln korrekt ist. Somit sind für die Prüfung eines implementierten *Translator-Plugins* die Verifikation und die Validierung relevant. Wenn nur die Validierung durchgeführt werden würde, könnte die Korrektheit der Spezifikation entschieden werden, obwohl deren Umsetzung fehlerhaft sein könnte. Wenn hingegen nur die Verifikation erfolgen würde, könnte eine fehlerhafte Spezifikation korrekt umgesetzt sein.

Bei der Verifikation und Validierung sind die Kriterien entscheidend, anhand derer die Transformation beurteilt wird. In der Literatur sind folgende Korrektheitskriterien zu finden:

- Syntaktische Korrektheit**:** Die Transformation erzeugt ein syntaktisch korrektes Modell oder sie ist syntaktisch korrekt hinsichtlich ihrer Sprache (vgl. Küster, 2004; vgl. Varro et al., 2003; vgl. Ehrig/Ehrig, 2006; vgl. Narayana, 2008; vgl. Lano/Kolahdouz, 2010).
- Syntaktische Vollständigkeit: Die Transformation übersetzt alle Elemente des Ausgangsmodells vollständig (vgl. Varro et al., 2003; vgl. Lano/Kolahdouz, 2010; vgl. Hermann et al., 2010).

- Terminierung**:** Die Transformation wird nach endlicher Zeit abgeschlossen (vgl. Narayana, 2008; vgl. Ehrig et al., 2005).
- Konfluenz, Einzigartigkeit**:** Die Transformation ergibt immer das gleiche Ergebnis, dabei ist die Anwendungsreihenfolge der Transformationsregeln nicht relevant (vgl. Küster, 2004; vgl. Varro et al., 2003; vgl. Ehrig et al., 2005; vgl. Lano/Kolahdouz, 2010).
- Semantische Korrektheit (Erhaltung des Verhaltens): Beim Zielmodell der Transformation bleiben die Semantik des Ausgangsmodells beziehungsweise wichtige semantische Eigenschaften erhalten (vgl. Küster, 2004; vgl. Varro et al., 2003; vgl. Ehrig/Ehrig, 2006; vgl. Ehrig et al., 2005;vgl. Narayana, 2008).

Die beiden Eigenschaften Konfluenz und Terminierung werden auch als funktionales Verhalten zusammengefasst (vgl. de Lara/Taentzer, 2004; vgl. Hermann et al., 2010).

5.2 Methoden der Verifikation und Validierung

Um Modelltransformationen zu verifizieren, existieren zwei grundlegend verschiedene Ansätze (vgl. Leitner, 2006). Zum einen können Transformationen auf Metamodelllevel, zum anderem nach jedem Ablauf verifiziert und validiert werden. Beim sogenannten *Checker Approach*, wird während jeder Transformation entschieden, ob Ausgangs- und Zielmodell semantisch äquivalent sind. Dieser Ansatz wird in der Literatur auch als *Modelllevel-Verifikation* (vgl. Varro, 2003) bezeichnet. Die *Metamodelllevel-Verifikation* (vgl. Varro, 2003) bzw. der regelbasierte Ansatz (vgl. Leitner, 2006) beweisen die allgemeine semantische Korrektheit für jede Transformation, bevor diese stattfindet.

Die folgenden Betrachtungen beschäftigen sich mit verschiedenen Ansätzen aus der Literatur, die auf Metamodelllevel oder Modelllevel Modelltransformationen verifizieren und validieren. Die Ansätze werden soweit ausgeführt, dass es möglich ist, deren Funktionsweise grob zu verstehen. Zudem werden sie für die Anwendbarkeit der Verifikation und Validierung für die Modelltransformationen dieser Arbeit überprüft. Die Betrachtungen basieren auf einer dieser Arbeit zugrundeliegenden Masterarbeit (Scharfe, 2011, S. 66 ff.).

5.2.1 Formale Verifikation durch Modelchecking

In Abbildung 5-1 wird die Transformation eines Ausgangsmodells A in ein Zielmodell B und die formale Verifikation der Transformation gezeigt. Dafür müssen die Ausgangs- und Zielsprache syntaktisch und semantisch genau beschrieben sein. Die Verifikation wird in folgenden Schritten durchlaufen:

1. Automatisierte Modellgenerierung**:** Für ein beliebiges, dem `Metamodell A` untergeordnetes Modell `Nutzer Modell A` wird durch das Transformationsprogramm `Nutzer Modell B` erzeugt.

2. Generierung des Transitionssystems: Für beide Modelle wird ein Zustandsübergangssystem automatisch generiert.

3. Auswahl der Korrektheitseigenschaften: Es wird eine Eigenschaft `p` in der `Ausgangssprache A` gewählt. Die Eigenschaft muss als Muster, bestehend aus Elementen und logischen Operatoren der Ausgangssprache, ausdrückbar sein.

4. Überprüfen des Ausgangsmodells: Die Eigenschaft p wird von einen Modell *Checker* Programm für das für A generierte Transitionssystem automatisch überprüft. Läuft die Überprüfung nicht erfolgreich ab, sind

    a. Inkonsistenzen im Ausgangsmodell vorhanden (Verifizierungsproblem),

    b. oder die informellen Anforderungen nicht ausreichend durch p abgebildet (Validierungsproblem),

    c. oder die formale Semantik der Sprache A ungeeignet (Validierungsproblem).

5. Transformation der Eigenschaft und Validierung der Transformation: Überführen der Eigenschaft p in die Zielsprache B. Da die Modelltransformation die Eigenschaft p fehlerhaft übertragen kann, sollten Experten die Korrektheit der Eigenschaft q validieren. Im Regelfall kann dieser Schritt nicht voll automatisiert werden.

6. Überprüfen des Zielmodells**:** Das für B erzeugte Transitionssystem wird gegen die Eigenschaft q geprüft. Wenn die Eigenschaft dem Transitionssystem genügt, ist die Transformation hinsichtlich des Paares (p,q) korrekt, ansonsten wurde die Eigenschaft p bei der Transformation nicht erhalten.

Die vorgestellte Methode zur formalen Verifikation von Modelltransformationen durch *Model Checking* ist halbautomatisch. Nach jeder Transformation werden Ausgangs- und Zielmodell in eine formale Semantik übertragen und von einem *Model Checker* verifiziert. Bei der Verifikation wird überprüft, ob Eigenschaften sowohl vor als auch nach der Transformation gelten. Bei der Methode ist es notwendig, das Ausgangs-, sowie auch das Zielmodell in eine formale Semantik zu überführen. Die semantisch formale und vollständige Beschreibung der Zielmodelle ist beim Untersuchungsgegenstand dieser Arbeit schwierig, da sie nicht niedergeschrieben und sehr komplex ist (vgl. Varro, 2003).

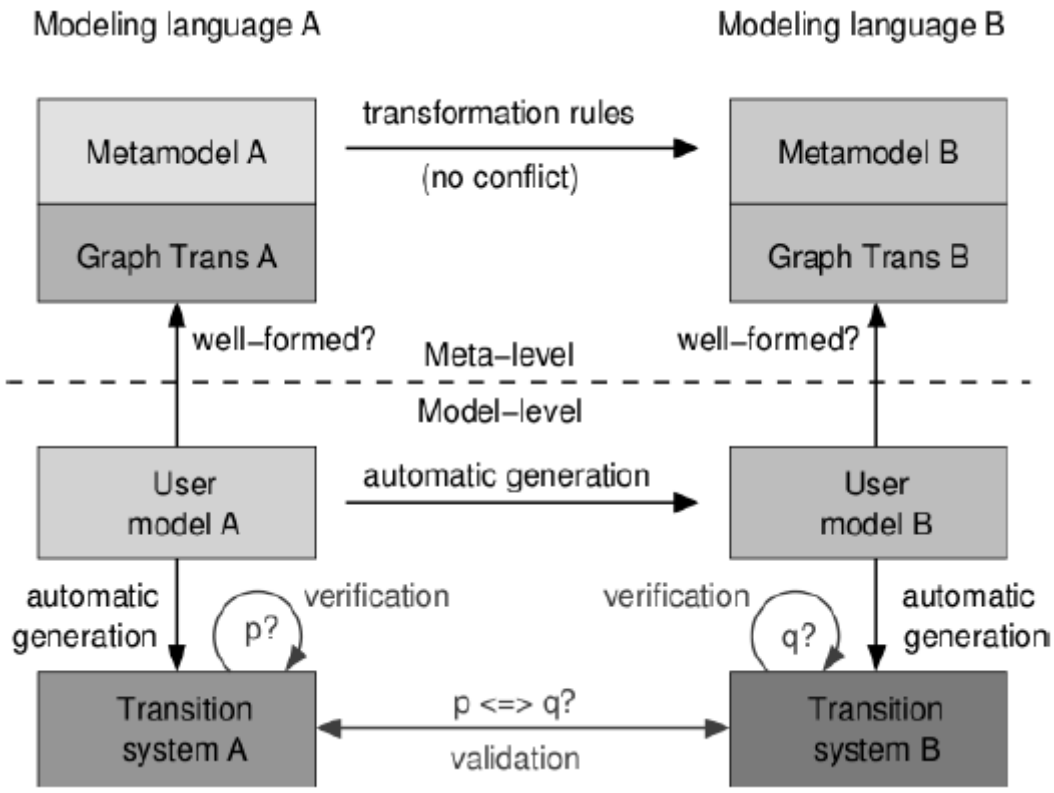


Abbildung 5-1: Checker Ansatz (vgl. Varro, 2003, S. 4)

5.2.2 Formale Verifikation durch Theorem-Beweiser

Während der *Checker*-Ansatz die Überprüfung konkreter Modellinstanzen formal verifiziert, verifiziert der regelbasierte Ansatz mit einem formalen Beweis direkt die Transformationsregeln. Beim regelbasierten Ansatz wird mit einen formalen Beweis erörtert, dass die Transformation eines Ausgangsmodells in ein Zielmodell dessen Semantik erhält. Dies gilt für jedes Ausgangsmodell, da die Regeln bewiesen werden (vgl. Leitner, 2006).

Der Ansatz arbeitet mit den vorgestellten *Triple Graph Grammars*. Wie im Kapitel 4.2.3 ausgeführt ist es möglich TGG-Regeln als Paar zusammenhängender Produktionen zu verstehen. So kann aus einer Konstruktionssequenz einer Modellinstanz eine Modellinstanz einer anderen Sprache parallel mitentwickelt werden. Somit sind zwei Modelle, die aus semantisch äquivalenten Metamodellen entstehen, bei der Anwendung gleicher TGG-Regeln wieder semantisch äquivalent. Als Grundlage müssen die Metamodelle der Ausgangs- und Zielsprache in eine formale Sprache überführt und mit Semantik ausgestattet werden. Die Definition der Semantik besteht aus drei Teilproblemen:

1. Definition einer semantischen Domäne, die es ermöglicht, formal die Bedeutung eines Modells darzustellen.
2. Definieren einer Abbildung vom Modelldatentyp auf die semantische Domäne.
3. Definieren einer Äquivalenzrelation, die es möglich macht, Modelle zu vergleichen.

Leitner verwendete dazu Isabelle, ein generischen interaktiven Theorembeweiser, der es ermöglicht Metamodelle zu beschreiben (vgl. Nipkow et. al., 2002). Um eine TGG-Regel beweisen zu können, ist es notwendig, sie zunächst in ihre einzelnen Graphen-Produktionen aufzuspalten. Anschließend werden die Produktionen formal abgebildet (beispielsweise als Funktionen nach Isabelle). Nun ist es notwendig, den Korrespondenzgraphen der TGG-Regeln zu formalisieren, indem Korrespondenzvoraussetzungen bestimmt werden, die den Zusammenhang der linken und rechten Seite ausdrücken. Anschließend ist es möglich, die Korrektheit der Regel mit dem Theorembeweiser interaktiv zu verifizieren.

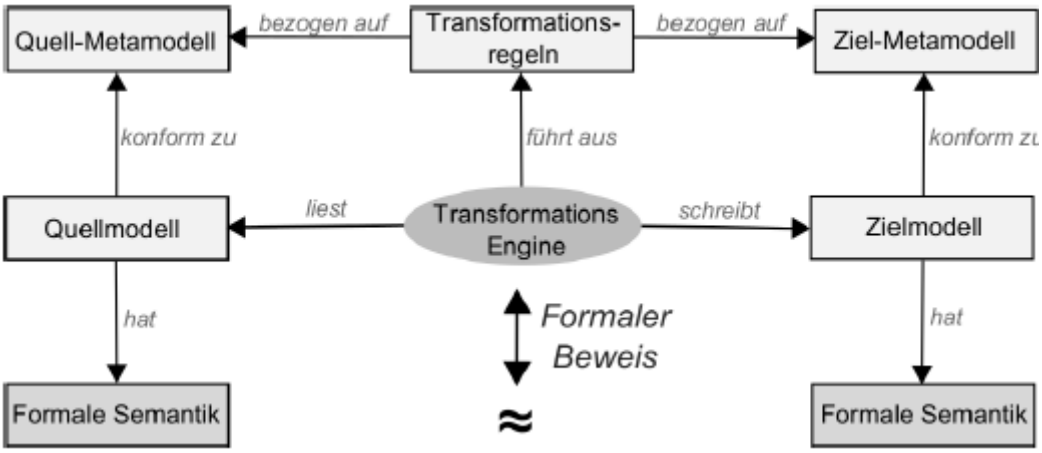


Abbildung 5-2: Metamodelllevel-Verifikation durch Theorem-Beweiser (vgl. Estler, 2009, S.3)

Der Vorteil der formalen Verifikation von Transformationsregeln gegenüber dem *Checker*-Ansatz ist, dass nur eine endliche Menge von Modellinstanzen geprüft wird und es möglich ist, mit dieser eine allgemeine Aussage über die Korrektheit einer Regel zu treffen. Da auch bei der formalen Verifikation die Semantik der Metamodelle formalisiert werden muss, ergeben sich allerdings dieselben Probleme wie beim *Checker*-Ansatz.

5.2.3 Verifikation durch Korrespondenzkriterien

In Narayana/Karsai (2008) wird eine Möglichkeit aufgezeigt, Modelltransformation anhand von sogenannten Korrespondenzkriterien zu validieren. Dafür nutzt Narayana die zur Spezifikation von Graphen entwickelte Sprache GReAT. Die Sprache erlaubt es, temporäre Kanten und Knoten zwischen den Metamodellen der Ausgangs- und Zielsprache zu spezifizieren. Diese werden als *Cross-Links* bezeichnet und ähneln den Korrespondenzgraphen der *Triple Graph Grammars*. Anhand der *Cross-Links* können Elemente von Modellinstanzen der Ausgangs- und Zielsprache in Beziehung gebracht und auf Erhalt von bestimmten Eigenschaften getestet werden. Dabei läuft die Transformation und Verifikation in folgenden vier Schritten ab:

1. Definition der Metamodelle der Ausgangs- und Zielsprache.
2. Definition der Transformationsregeln und der Regeln für die strukturellen Korrespondenzen.
3. Automatisierte Zielmodellgenerierung.
4. Verifikation der Transformation durch Auswerten der Korrespondenzregeln.

Korrespondenzregeln beschreiben, wie ein Merkmal des Ausgangsmodells im Zielmodell umgesetzt wird. Für jedes Merkmalpaar kann eine Menge von Korrespondenzregeln definiert werden. Um die Modelle nicht vollständig durchlaufen zu müssen, werden die Metamodelle der Ausgangs- und Zielsprache durch sogenannte *Cross-Links* verbunden. Nach der Transformation erfolgt die Auswertung der Korrespondenzregeln (vgl. Abbildung 5-3). Dazu wird eine Tiefensuche durchgeführt und bei jeder Korrespondenz wird geprüft, ob die Eigenschaften der Korrespondenzregel genügen. Die Überprüfung kann automatisch erfolgen. Der Ansatz eignet sich gut, um strukturelle Eigenschaften von Modellen zu überprüfen.

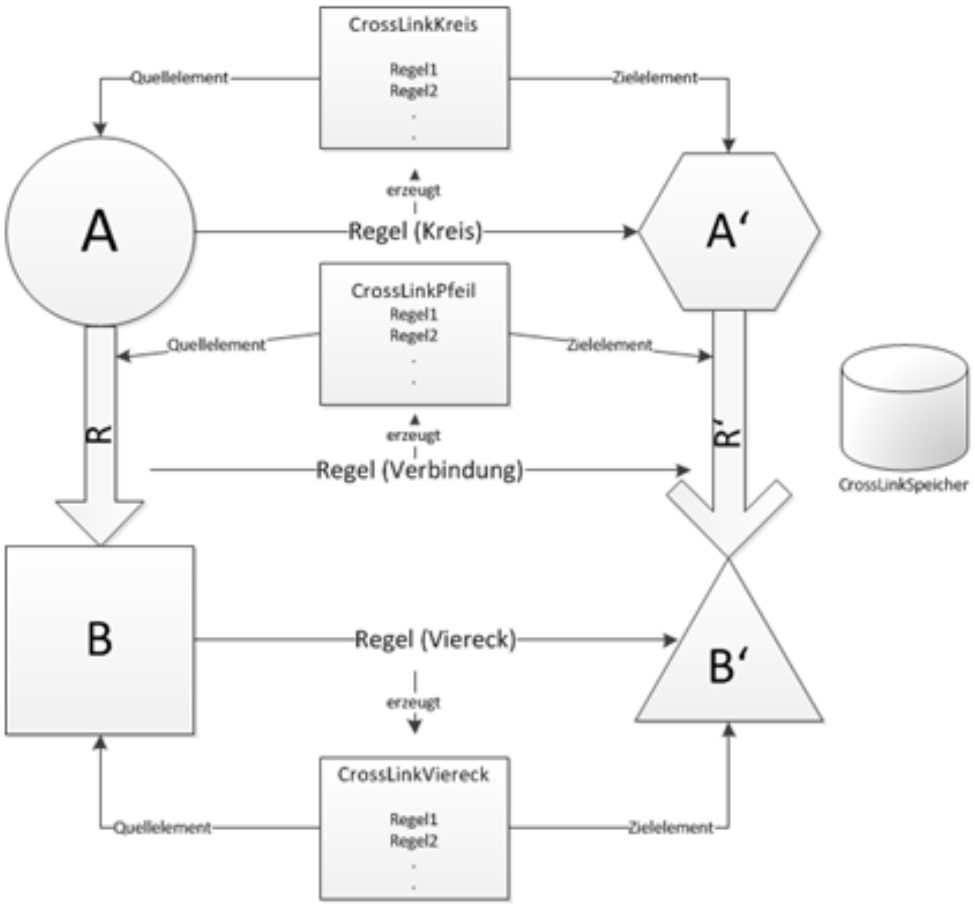


Abbildung 5-3: Verifikation durch strukturelle Korrespondenzen (vgl. Winter, 2012, S.60)

5.2.4 Verifikation durch Abhängigkeitsgraphen

Küster stellt die Methode Verifikation durch Abhängigkeitsgraphen für die Verifikation auf nicht formaler Basis vor (vgl. Küster, 2004). In den Ausarbeitungen werden die Modelle durch Graphgrammatiken beschrieben, die den im Kapitel 4.2.3 beschriebenen *Pair Graph Grammars* ähneln. Durch den Ansatz ist es möglich, Graphtransformationen syntaktisch zu beschreiben (vgl. Abbildung 5-4). Die Methode überprüft den Aufbau eines Abhängigkeitsgraphen, auf die Erzeugung von Nicht-Terminalen. So ist es möglich, Regeln zu ermitteln, bei deren Anwendung nicht-korrekte Zielmodelle erzeugt werden. Die Fehlerfreiheit des Graphen hinsichtlich dieser Prüfung ist aber nur eine notwendige Bedingung für die Korrektheit der Syntax, da es bei der Erzeugung nicht möglich ist sicherzustellen, ob eine Regel, die eine Nicht-Terminale löscht, auch wirklich angewendet werden kann. Abhängigkeitsgraphen gehen mit einer sehr komplexen Graphtransformation einher, was gerade bei der Transformation großer Modelle sehr aufwendig ist. Bei komplexen Sachverhalten können die resultierenden Transformationsregeln schnell unübersichtlich werden. Dies kann den Vorteil der syntaktischen Verifikation wieder nivellieren.

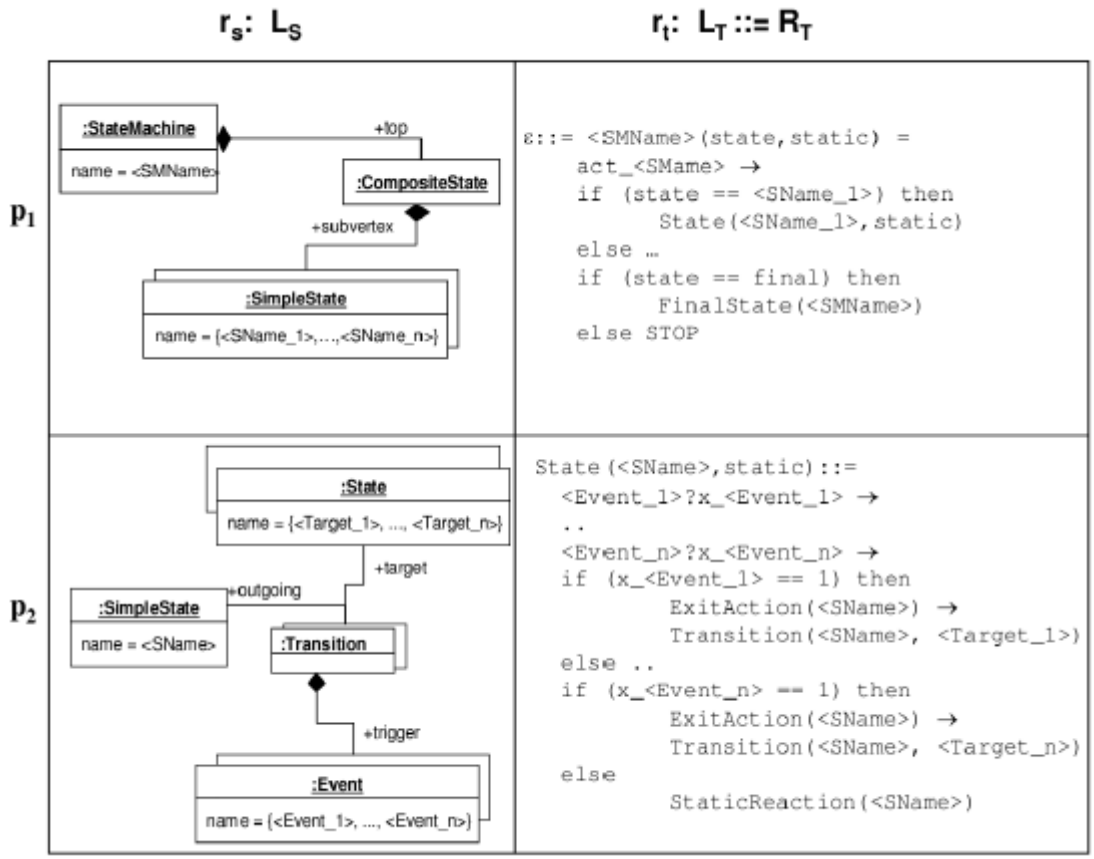


Abbildung 5-4: Beispiel Transformationsregeln für UML Zustandsdiagramme (Küster, 2004, S. 4)

5.2.5 Validierung durch Testen

Validieren durch Testen ist eine weitverbreitete Methode, die beispielsweise in der Softwaretechnik verwendet wird (vgl. Cleff, 2010). Dabei wird anhand einer Menge von Testfällen geprüft, ob die Ergebnisse Erwartungen entsprechen. Baundry (2009) zeigt Möglichkeiten zum systematischen Testen von Modelltransformationen. Dabei werden folgende Aktivitäten vorgestellt, die für den Test einer Transformation notwendig sind:

1. Generierung von Testdaten: Es erfolgt die Generierung von Testmodellen, die dem Metamodell der Ausgangssprache genügen.
2. Festlegung von Testkriterien: Da es aus Zeitgründen nicht möglich ist, alle Modelle des Ausgangsmodells bei der Transformation zu testen, werden Testkriterien zur sinnvollen Erzeugung der Testmodelle ausgewählt.
3. Erstellung eines Orakels: Bei der automatischen Ausführung des Testens ist es notwendig, ein Programm zu entwickelt, das über die Korrektheit eines Testfalles entscheidet. Dieses Programm wird als Orakel bezeichnet.

Die automatische Generierung von Testfällen kann gerade bei sehr komplexen Modellen mit zahlreichen *Constraints* sehr rechenintensiv sein. Zudem kann auch die Erstellung eines solchen Programmes, mit entsprechenden Kriterien und *Constraints* für sinnvolle Testmodelle sehr zeitaufwendig sein. Das Orakel kann das Zielmodell durch verschiedene Möglichkeiten testen. Eine Möglichkeit besteht darin, das Zielmodell, soweit es ausführbar ist, direkt zu testen und wenn möglich mit einer Ausführung des Ausgangsmodelles zu vergleichen. Ein anderer Ansatz ist das Testen einzelner Eigenschaften. In der Softwaretechnik wird dies beispielsweise mit *Unit Tests* durchgeführt (vgl. Osherove, 2010).

Beim Testen werden zwei Probleme deutlich: zum einen das Problem der Auswertung durch ein Orakelprogramm und zum anderem das Problem der Testfallgenerierung. Für die Transformation von SysML-Modellen in Simulationsmodelle gestaltet sich die Validierung allein durch Testen aufwändig, da für Ausgangs- und Zielmodelle Simulationsergebnisse generiert werden müssen. Durch das automatische Erstellen von Modellen nach ausgewählten Kriterien lässt sich der Aufwand minimieren.

5.2.6 Verifikation und Validierung durch Invarianten

In Cabot et al. (2010) wird eine Methode für die Verifikation und Validierung durch Invarianten vorgestellt. Ein Transformationssystem besteht dabei aus dem Metamodell der Ausgangs- und Zielsprache, sowie einer Menge von auf der *Object Constraint Language* (OCL) aufbauender Invarianten. Invarianten sind Zusicherungen für Assoziationen oder Instanzen, im Transformationsmodell sind sie die Regeln der Modelltransformation.

Das Werkzeug UMLtoCSP ermöglicht es, UML-Transformationsmodelle automatisch in ein *Constraint Satisfaction Problem* (CSP) zu überführen. CSPs bestehen aus Variablen, Domänen sowie Zusicherungen und ermöglichen es, durch Invarianten Eigenschaften zu verifizieren. Dazu ist es notwendig, die Modelle mit einem UML-Modellierungswerkzeug zu entwerfen, das auf dem Ausgabeformat XMI basiert. Zusätzlich müssen die Transformationsregeln in OCL-Invarianten umgewandelt und in eine Text-Datei abgelegt werden. Wie Transformationsregeln in OCL-Invarianten umgewandelt werden, beschreibt Cabot ausführlich (vgl. Cabot et al., 2010). Nun ist es möglich, bestimmte Eigenschaften anhand der (Nicht-)Existenz von Lösungen zu verifizieren. Der Nutzer bestimmt den Suchraum durch die Wahl von zu prüfenden Variablen und deren Wertebereich. Das Werkzeug generiert automatisch Lösungen bis ein Gegenbeispiel gefunden wurde oder der Suchraum vollständig durchlaufen wurde. Durch den eingeschränkten Suchraum terminiert jede Transformation, doch wurde auch nur ein Teil des gesamten Suchraumes überprüft. Über den Suchraum lassen sich auch funktionale Kriterien überprüfen, wie etwa die Eindeutigkeit und die Terminierung der Modelltransformation.

Es ist nicht möglich, die Validierung ohne Einfluss des Anwenders durchzuführen. Cabot et al. beschreiben, dass mit Hilfe der Invarianten für Ausgangsmodelle Zielmodelle oder für Zielmodelle Ausgangsmodelle erstellt werden können, die der Anwender dann manuell validiert. Der Vorteil gegen-

über der einfachen Validierung von transformierten Modellen ist, verifizierte Modelle zu validieren. Wenn mit anderen Methoden Modelle erst verifiziert und dann manuell validiert werden, sind die Voraussetzungen jedoch ähnlich.

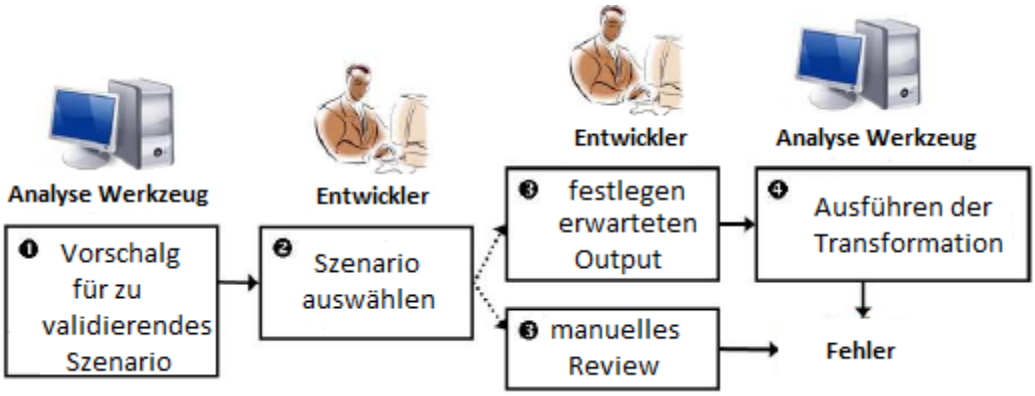


Abbildung 5-5: Validierung von Modelltransformationen mit Hilfe von Analyse-Tools (Cabot et al., 2010, S. 40)

Die angegebenen formalen Beschreibungen für die Umwandlung der Metamodelle und Transformationsregeln in ein CSP ermöglichen es, den Ansatz auch auf Anwendungsfälle außerhalb der UML zu übertragen. Da das Werkzeug UMLtoCSP jedoch kein SysML behandeln kann, wären für die Nutzung des Ansatzes für die Transformationen dieser Arbeit folgende Schritte notwendig:

1. Auswahl oder Erstellung eines geeigneten Werkzeuges.
2. Detaillierte Spezifikation des SysML-Metamodells und seiner *Constraints*.
3. Detaillierte Spezifikation des jeweiligen Ziel-Metamodells und seiner *Constraints*.
4. Überführung der imperativen Transformationsregeln in OCL Invarianten (vgl. Abbildung 5-5).

Die Umsetzung dieser Schritte scheint sehr aufwändig zu sein. Schon allein die Umsetzung des SysML-Metamodells und seiner *Constraints* ist mit viel Arbeit und Fehleranfälligkeit behaftet.

5.2.7 Auswertung der Eignung der Methoden

In den vorherigen Abschnitten wurden verschiedene Ansätze für die Verifikation und Validierung von Modelltransformationen vorgestellt und im Kontext dieser Arbeit bewertet. Es zeigt sich, dass sich die vollständige automatisierte Validierung der semantischen Korrektheit kaum umsetzen lässt. Zum einen ist es schwierig, die spezifischen Metamodelle der Simulatoren zu erfassen, da sie oft nur sehr ungenau als Benutzerhandbuch vorhanden sind. Zum anderen besteht ein Mangel an Beschreibungsmöglichkeiten von solch komplexen Semantiken.

Im Abschnitt 5.2.3 wird eine Möglichkeit zur Überprüfung der Syntax und Semantik mit Hilfe von Korrespondenzkriterien gegeben. Sie ermöglichen es, bestimmte Beziehungen zwischen Ausgangs-

und Zielmodell zu spezifizieren, die nach einer Transformation automatisch verifiziert werden können. Der Ansatz überprüft nur bestimmte Kriterien, die der Nutzer als Korrespondenzregeln definiert. Die Auswahl der Kriterien ist daher sorgfältig zu entscheiden. Zudem erfolgt die Überprüfung nicht auf Metamodelllevel, sondern nach jeder einzelnen Transformation, was durch die Definition einer Menge geeigneter Testszenarien kompensiert werden kann. Jedoch ist es so möglich, syntaktische und semantische Überprüfungen gleichermaßen und mit überschaubarem Aufwand umzusetzen. Dieser Ansatz scheint auch für die Umsetzung innerhalb der implementierten *Translatorplugins* geeignet. Dafür müssen geeignete Korrespondenzkriterien definiert und das Programm erweitert werden.

Eine weitere Methode zur Überprüfung der Semantik ist das im Abschnitt 5.2.5 beschriebene Testen. Grundlage ist, dass das Ausgangs- sowie Zielmodell zum Testen zur Verfügung stehen. Wenn eines der beiden Modelle genau definiert ist, besteht die Möglichkeit, das Äquivalent der Transformation auch auf deterministische Fälle zu testen.

Im Abschnitt 5.2.4 wird gezeigt, wie sich die Syntax einer Modelltransformation mit der Hilfe von Abhängigkeitsgraphen überprüfen lässt. Dabei wird mit Hilfe der Abhängigkeitsgraphen getestet, ob Regeln existieren, die ungültige Zielmodelle erstellen können. Um die Umsetzbarkeit des Ansatzes bewerten zu können, ist es notwendig, festzustellen, ob sich die durchgeführten Transformationen auf diese Art sinnvoll beschreiben lassen. Aller Voraussicht nach würden die entstehenden Regeln für komplexe Probleme eher unübersichtlich werden, was die gewonnenen Vorteile aufheben könnte. Nicht zuletzt stellt die Überprüfung durch Abhängigkeitsgraphen, wie beschrieben, nur eine notwendige Bedingung dar.

Die Verifikation und Validierung durch Invarianten bietet eine ganze Reihe von Möglichkeiten. So ist es möglich, neben den syntaktischen Eigenschaften auch die funktionalen Eigenschaften zu überprüfen. Jedoch arbeitet die Methode Probleme mit Spezifikation der Variablen sequenziell ab. Es können vom Nutzer definierte Suchräume geprüft werden, es ist aber nicht möglich, große und komplexe Probleme vollständig in einer akzeptablen Zeit zu überprüfen. Zudem ist – wie beschrieben – ein erhöhter Aufwand durch die Entwicklung eines geeigneten Werkzeuges zu erwarten.

5.3 Annahmen für die Verifikation und Validierung der Modelltransformation

Innerhalb des im Kapitel 4 vorgestellten Transformationssystems finden mehrere aufeinander aufbauende Schritte statt, die jeweils eine eigene Betrachtung bei der Verifikation und Validierung benötigen (vgl. Abbildung 5-6). Im ersten Schritt erstellt der Anwender ein Szenario, das den Anforderungen des Metamodells genügen muss. Im zweiten Schritt wird das Modell vom Modellierungstool in das interne Java Modell übertragen. Im dritten Schritt wird das Modell dann für ein bestimmtes Modellierungstool

aufbereitet. Die folgenden Abschnitte sind anhand dieser Dreiteilung untergliedert und beschreiben jeweils zugehörige Betrachtungen im Kontext der Verifikation und Validierung.

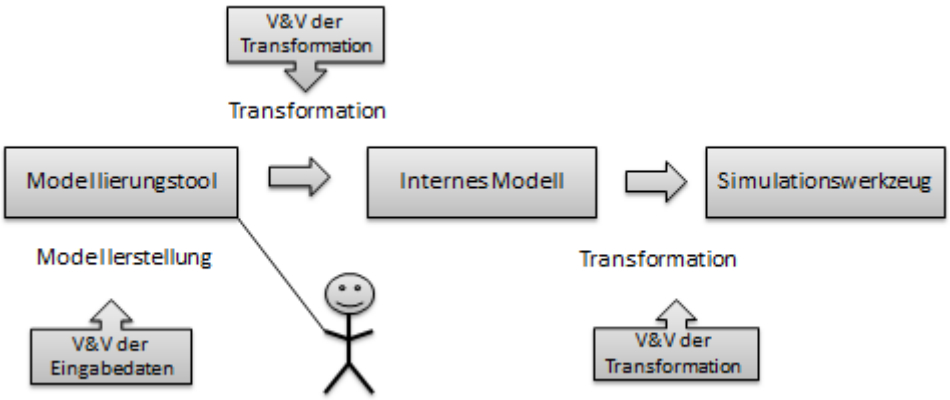


Abbildung 5-6: Schnitstellen für die Verifikation und Validierung des Gesamtmodells

5.3.1 Verifikation und Validierung bei der Modellerstellung

Wenn ein Anwender ein Szenario mit dem Konzept der Arbeit erstellen und übersetzen möchte, kann er dafür MagicDraw mit einen speziellen Profil oder das in dieser Arbeit entwickelte TOPCASED *Engineer* nutzen. Der Nutzer arbeitet nun auf dem Metamodell dieser Arbeit (vgl. Kapitel 3) und bekommt daher durch das genutzte Profil nur Sprachelemente angeboten, die im Metamodell enthalten sind. Der Anwender kann mit den zur Verfügung stehenden Sprachelementen jedoch Modelle mit syntaktischen oder semantischen Fehlern erzeugen. Häufige syntaktische Fehler sind beispielsweise Elemente in einem Szenario in Relation zu bringen, ohne dass für sie Relationen im Metamodell definiert sind (beispielsweise Prozesse mit Warteschlangen). Zudem ist es möglich, Variablen falsch (beispielsweise Zusammensetzung von Verteilungen) oder außerhalb ihrer Wertebereiche zu definieren. Eine weitere Fehlerquelle beim Modellieren sind semantisch falsch modellierte Sachverhalte (beispielsweise zehn benötigte Arbeiter aber nur drei im Szenario). Die syntaktischen Fehler führen zu einem Abbruch bei deren Überprüfung. Bei den semantischen Fehlern können die Modelle in ein Simulationsmodell transformiert werden, führen dann aber zu Fehlern bei der Ausführung des Simulationsmodells.

Diese Möglichkeit der Fehlmodellbildung kann durch verschiedene Mechanismen unterbunden werden. So ist es möglich, die Modelle während oder nach ihrer Konstruktion zu überprüfen. Das Modellierungsprogramm TOPCASED lässt es nur zu, Modelle zu entwerfen, die den syntaktischen Anforderungen von SysML genügen. Doch werden die Anforderungen in dieser Arbeit durch die Verwendung des speziellen Modellierungskonzeptes zusätzlich eingeschränkt. Die Überwachung dieser Anforderungen zur Laufzeit würden es erfordern, den TOPCASED Quellcode anzupassen, was einen erheblichen Arbeitsaufwand erfordern würde. Daher wird die syntaktische Überprüfung auf Einhaltung des

Metamodells nach der Erstellung des Zielmodells durchgeführt. Wenn der Anwender das Modell exportiert, wird es mit Hilfe von Regeln auf die Einhaltung der Syntax des Metamodells geprüft. Die automatische Prüfung eines eingelesenen Modells auf die Einhaltung von Regeln ist ohne größeren Aufwand umzusetzen. Nach der Überprüfung erhält der Anwender eine Meldung mit Fehlmodellierungen. Das Modell wird beispielsweise auf Fehlrelationen, verwaiste Elemente und falsche Variablendeklaration untersucht. Überprüfungen auf semantische Fehler sind ebenfalls mit einfachen Regeln möglich. Die Umsetzung benötigt lediglich einen domänenspezifischen Regelkatalog, weshalb sie ohne größeren Aufwand erfolgen kann.

5.3.2 Verifikation und Validierung bei der Transformation in das interne Modell

In einer ersten Transformation wird das erstellte Modell vom Modellierungswerkzeug in ein internes Java-Datenmodell transformiert. Wie im Kapitel 4.3.1 beschrieben, ist das Java-Datenmodell eine Abbildung des verwendeten Metamodells des Modellierungskonzeptes. Die Daten werden lediglich vom unübersichtlichen und informationsredundanten XMI-Ausgabeformat in das übersichtliche Äquivalent eines Java-Datenmodells übertragen. Da Ausgangs- und Zielmodell auf dem gleichen Metamodell basieren, ist die Fehleranfälligkeit dieses Transformationsschrittes als gering zu bewerten. Die syntaktische Korrektheit, semantische Korrektheit und syntaktische Vollständigkeit wurde durch eine große Zahl von Testkriterien überprüft (vgl. Partzsch 2010).

5.3.3 Verifikation und Validierung bei der Transformation in das jeweilige Simulationswerkzeug

Die Transformation vom internen Java Modell zu einem speziellen Simulationswerkzeug bildet zwischen zwei verschiedenen Metamodelle ab. Die Fehleranfälligkeit ist daher weitaus größer als bei der Erzeugung des internen Modells und wird im folgenden Abschnitt genau betrachtet.

5.4 Verifikation und Validierung der Modelltransformation anhand von Korrektheitskriterien

Die folgenden Ausführungen richten sich nach den im Kapitel 5.1 angegebenen Korrektheitskriterien für die Verifikation und Validierung von Modelltransformationen. Die Betrachtungen stehen speziell im Kontext der in dieser Arbeit durchgeführten Übersetzungen, um ein Konzept für deren Verifikation und Validierung zu erstellen. Es ist möglich, Modelltransformation als Graphen-Transformation zu betrachten, wenn der ursprüngliche Graph durch Anwendung von Regeln durch einen anderen Graphen ersetzt wird und in der Ausgangssprache keine Knoten ersetzt, erzeugt oder entfernt werden (vgl. Küster et al., 2006; vgl. Heckel et al., 2002). Da die Implementierungen dieser Anforderung genügen, ist es möglich, sie im Folgenden auch als Graphentransformation zu betrachten. In zugrundeliegenden

Arbeiten wurde die Verifizierung und Validation der Übersetzung von SysML nach Factory Explorer (vgl. Scharfe, 2011) und nach Flexsim durchgeführt (vgl. Winter, 2012). Die Ergebnisse werden als Referenz verwendet, um eine geeignete Methodik zur Verifikation und Validierung für die Modelltransformationen zu ermitteln.

5.4.1 Terminierung

Wie beschrieben, terminiert die Transformation, wenn sie nach endlicher Zeit abgeschlossen ist. Ob ein Graphentransformationssystem jedes Mal terminiert, ist jedoch nicht immer entscheidbar, da es nicht immer möglich ist zu beweisen, dass eine nichtterminierende Graphentransformation nicht terminiert (Plump, 1998, S. 3 f.). Die Problematik ist dem Halteproblem von Turingmaschinen ähnlich (Chaitin, 2007, S. 156). Beispiele für das Nichtdeterminieren wären das Auftreten von unendlichen Iterationen oder *Deadlocks*. Jedoch ist es möglich zu zeigen, dass eine Graphen-Transformation terminiert, indem nachgewiesen wird, dass für jeden Transformationsschritt eine Reduktion der Gesamtschritte oder der Gesamtzeit erfolgt. Dies kann nachgewiesen werden, indem die Transformation auf ein Reduktionssystem zurückgeführt wird (Küster et al., 2004, S. 5). Diese Art der Beweisführung ist lediglich formal oder semi-formal möglich, was die Validität der Aussage senkt (vgl. Plump, 1998).

Die Überprüfung auf Terminierung der Übersetzer vom Factory Explorer und Flexsim wurden mittels semiformaler Beweisführung durchgeführt. In beiden Fällen führte die Überprüfung zum Ergebnis, dass die beiden Übersetzer terminieren (Scharfe, 2011, S. 99 ff.; Winter, 2012, S. 8 ff.). Anhand der semi-formalen Durchführung der Beweise ist deren Aussagekraft jedoch gemindert.

5.4.2 Konfluenz

Die Konfluenz sagt aus, ob eine Transformation immer zum gleichen Ergebnis führt, wobei die Anwendungsreihenfolge der Transformationsregeln unabhängig ist (vgl. Küster, 2004; vgl. Varro et al., 2003; vgl. Ehrig et al., 2005; vgl. Lano/Kolahdouz, 2010). Für jedes Ausgangsmodell soll genau ein Zielmodell erzeugt werden. Bei der Konfluenz wird jedoch nicht gefordert, dass semantisch gleiche Ausgangsmodelle semantisch äquivalente Zielmodelle ergeben. So ist es auch möglich, eine Transformation als konfluent zu werten, wenn zwei semantisch äquivalente Ausgangsmodelle in zwei semantisch nicht äquivalente Zielmodelle transformiert werden, solange zu jedem Zeitpunkt dieselben nicht äquivalenten Modelle erzeugt werden (vgl. Küster, 2004). Die praktischen Untersuchungen haben gezeigt, dass es notwendig ist, die Modellgleichheit differenzierter zu betrachten. Die Bewertung der Transformationen dieser Arbeit erfolgt durch die folgende Skala:

- Absolute Identität: Bei der Eingabe eines Ausgangsmodells wird immer dasselbe Zielmodell erstellt. Die nach verschiedenen Durchläufen entstehenden Modelle sind syntaktisch äquivalent.

- Nahe Identität: Bei mehreren Durchläufen erhalten die Elemente des Zielmodells unterschiedliche IDs, ansonsten sind die Modelle syntaktisch äquivalent.
- Strukturelle Identität: Die Modelle unterscheiden sich neben den IDs in der Reihenfolge in der die Elemente abgebildet werden, ansonsten sind die Modelle syntaktisch äquivalent.
- Semantische Gleichheit: Die Zielmodelle sind semantisch, aber nicht syntaktisch äquivalent.
- Ungleich: Die Modelle sind weder syntaktisch noch semantisch äquivalent.

Heckel beschreibt eine Möglichkeit, um Modelltransformationen auf Konfluenz zu testen. Dazu definiert er folgende Eigenschaften von Modelltransformationen (Heckel et al., 2002, S. 7 ff):

- Parallele Unabhängigkeit: Transformationsschritte können in beliebiger Reihenfolge angewandt werden und führen immer zum gleichen Resultat.
- Church-Rosser-Eigenschaft/-Theorem: Sind alle ausführbaren Regeln eines Systems parallel unabhängig, so ist es konfluent.
- Kritische Paare: Zwei Transformationsschritte, die nicht parallel unabhängig sind, werden als kritisches Paar bezeichnet.
- Kombinierbarkeit kritischer Paare: Obwohl die Regeln nicht parallel unabhängig sind, beeinflussen deren Reihenfolgeänderungen nicht die Semantik des Zielmodells.
- Kritische-Paare-Lemma: Sind alle kritischen Paare kombinierbar, ist das System konfluent.

Um zu testen, welche Transformationsschritte kritische Paare sind, und ob sie kombinierbar sind, wurden die Transformationsregeln aus dem Quellcode extrahiert, analysiert und beim Verdacht einer gegenseitigen Beeinflussung getestet. Zudem wurden die verschiedenen Testfälle transformiert, um deren Ergebnisse zu vergleichen.

Beim Flexsim-Export erfolgt die Generierung der Namen der Elemente verschieden (mit zufälligen Suffix), wodurch es nicht möglich ist, die Transformation als absolut identisch zu werten. Des Weiteren wurden zwei kritische Paare gefunden. Bei der Übersetzung eines `DecisionNode`, werden seine Nachfolgerknoten aus dem Datentyp `HashMap` des Java Standarteditors über einen Java-Iterator bestimmt. Eine Java `HashMap` ist vergleichbar mit dem algebraischen Konstrukt einer Menge. Die Entnahme der Elemente erfolgt daher nicht-deterministisch (vgl. Sun Microsystes, 2013).

Ausgangsmodelle können aus mehreren Aktivitätsdiagrammen bestehen, die hierarchisch gekapselt sind. Wenn zwei Aktivitätsdiagramme verschiedener Hierarchie übersetzt werden sollen, ist die Reihenfolge der Auswahl zufällig. Auch bei diesem kritischen Paar ist dies durch den genutzten Java Datentyp `HashMap` bedingt. Diese nicht-deterministische Auswahl hat die Generierung von IDs mit unterschiedlicher Reihenfolge bei der Übersetzung gleicher Modelle zur Folge. Doch wird weder die Semantik des Szenarios, noch die Reihenfolge der Elemente durch das kombinierbare kritische Paar beeinflusst (vgl. Winter, 2012, S. 62 f.). Bei der Transformation von `DecisionNodes` des Zielmo-

dells erfolgt die Generierung von direkten nachfolgenden Knoten daher in zufälliger Reihenfolge. In der Semantik werden keine Ungleichheiten generiert, doch die Elemente erhalten durch ihre verschiedenen Generierungsreihenfolgen in verschiedenen Transformationen verschiedene IDs und Reihenfolgen, mit denen sie abgebildet werden. Daher wurde die Transformation als strukturell identisch bewertet.

Auch bei der Verifikation und Validierung des Factory Explorers konnten kritische Paare gefunden werden, die sich jedoch alle nicht beeinflussten und daher als kombinierbar gewertet werden können (Scharfe, 2011, S. 99ff.). Alle gefundenen Paare, die bei der Verifikation und Validierung des Factory Explorers und von Flexsim als nicht parallel unabhängig klassifiziert wurden, basieren auf der Nutzung des nicht-deterministischen Java Datentyps `HashMap`. In der *Java Language Specification* sind verschiedene deterministische und nicht-deterministische Datenstrukturen festgelegt. Dabei muss dem nicht-deterministischen Datentyp `java.util.Set` besondere Beachtung geschenkt werden. Die einzige auf `java.util.Set` implementierte Datenstruktur, die eine interne Ordnung aufweist, ist `java.util.LinkedHashSet`. Alle anderen Datenstrukturen, die auf `java.util.Set` aufbauen, haben keine interne Ordnung und bringen das Risiko inkonfluenter Übersetzungen mit sich (vgl. Sun Microsystems, 2013).

5.4.3 Syntaktische Korrektheit und syntaktische Vollständigkeit

Bei der syntaktischen Korrektheit wird geprüft, ob die Transformation ein syntaktisch korrektes Modell erzeugt oder das Zielmodell hinsichtlich seiner Sprache korrekt ist (vgl. Küster, 2004; vgl. Varro et al., 2003; vgl. Ehrig/Ehrig, 2006; vgl. Narayana et. al., 2008; vgl. Lano/Kolahdouz, 2010). Jedes Modell, das transformiert wird, muss also den syntaktischen Bedingungen der Zielbeschreibungssprache genügen. Der Beweis der syntaktischen Korrektheit auf Metamodelllevel ist sehr umfangreich und benötigt grundlegend die vollständige syntaktische Definition der Metamodelle von der Ausgangs- und Zielsprache sowie die formale Beschreibung aller Transformationsregeln. Anhand der Betrachtungen aus Kapitel 5.2, wurde die Methode der Korrespondenzkriterien für die Überprüfung der syntaktischen Korrektheit und Vollständigkeit gewählt (vgl. Kapitel 5.2.7).

Die syntaktische Vollständigkeit einer Transformation ist gegeben, wenn alle Elemente und deren Eigenschaften des Ausgangsmodells vollständig abgedeckt sind (vgl. Varro et al., 2003; vgl. Lano/Kolahdouz, 2010; vgl. Hermann et al., 2010). Dabei ist zwischen Elementen und Eigenschaften zu unterscheiden, deren Umsetzung im Zielmodell möglich ist und deren Umsetzung im Zielmodell nicht möglich ist. Um dies auf dem Metamodelllevel zu beweisen, muss gezeigt werden, dass für jedes Element des Ausgangsmodells eine Transformationsregel vorhanden ist und jedes Ausgangsmodell während einer Transformation vollständig durchlaufen wird. Da auch dieser Nachweis sehr komplex ist, wurde die Überprüfung der syntaktischen Vollständigkeit in dieser Arbeit ebenso auf Modellebene mit Korrespondenzkriterien durchgeführt.

Um die syntaktische Korrektheit und Vollständigkeit zu verifizieren und validieren, wurden Korrespondenzkriterien definiert, die in einer Ansammlung von Testmodellen geprüft wurden. Eine ausreichende Menge an Testszenarien, die alle Wechselwirkungen der Elemente berücksichtigt, ist durch die Komplexität des Metamodells kaum möglich. So wurden Testkriterien definiert, die das Metamodell möglichst vollständig in seinen Ausprägungen abdecken. Insgesamt wurden 54 Korrespondenzkriterien für das statische Modell, 38 Korrespondenzkriterien für das dynamische Modell und 32 Testszenarien definiert (vgl. Winter, 2012).

Beim *Flexsim Exporter* werden die Eigenschaften `batchSize`, `priority`, `modePriority` und `capacity` der jeweiligen Elemente nicht übersetzt. Trotzdem ist deren Umsetzung in Flexsim möglich. Daher sind diese Eigenschaften als Mangel der syntaktischen Vollständigkeit aufzuführen. Weiterhin ist es nicht möglich, die Eigenschaften der `ExtendedAssociation` beim *Flexsim-Exporter* zu übersetzen (Winter, 2012, S. 29). Beim Übersetzer des Factory Explorers war es nicht möglich, die Eigenschaften `failure` der `Queue` sowie des `SingleProcess`, `qualification` der `Worker`, `capacity` sowie `speed` des Elements `Transporter` und die Eigenschaften der `ExtendedAssociation` abzubilden(Scharfe, 2011, S. 28ff.).

Fehler der syntaktischen Korrektheit konnten bei keinem der beiden Programme festgestellt werden.

5.4.4 Semantische Korrektheit

Semantische Korrektheit bedeutet, dass beim Zielmodell der Transformation die Semantik des Ausgangsmodells beziehungsweise wichtige semantische Eigenschaften, erhalten bleiben (vgl. Küster, 2004; vgl. Varro et al., 2003; vgl. Ehrig et al., 2005; vgl. Narayana, 2008). Um die Semantik der Modelle zu testen, ist es sinnvoll, die einzelnen Transformationsregeln direkt auf die semantischen Eigenheiten zu untersuchen. Dies kann durch Vergleiche von Simulationsabläufen des Ausgangs- und Zielmodells durchgeführt werden. Hierbei ist eine Ansammlung von Beispielen, die zu überprüfende Kriterien abdecken, sinnvoll. Des Weiteren sind Tests von Regelkombinationen notwendig (Rabe et al., 2006, S. 59 f.).

Die Aussagen über die Korrektheit der übersetzten Semantik sind wie bei der Syntax differenziert zu betrachten. Für die Auswertung der Semantik bei der Verifikation und Validierung von Flexsim wurde eine vierstufige ordinale Skala gebildet:

- Erfüllt**:** Es war möglich, das Element ohne semantische Abweichungen zu transformieren.
- Erfüllt unter Einschränkungen: Das Element wurde unter dem vom Zielmodell zur Verfügung stehenden Mitteln korrekt transformiert. Jedoch sind im Zielmodell die zur Verfügung stehenden Mittel nicht ausreichend, um die Semantik vollständig korrekt darzustellen (d. h., es gibt semantische Abweichungen, jedoch ist die Semantik im Zielmodell wiedererkennbar). Da im Zielmodell keine Mittel zur Verfügung stehen, das zu übersetzende Modell vollständig semantisch gleich darzustellen, können keine Verbesserungen durchgeführt werden.

- Erfüllt unter Bearbeitung: Das Element wurde mit semantischen Abweichungen transformiert. Das Zielmodell bietet jedoch die Möglichkeiten das Element semantisch vollständig korrekt darzustellen. Die semantischen Abweichungen können somit durch Abänderung der Transformationsregel behoben werden.

- Nicht erfüllt: Die Semantik des übersetzten Elements entspricht nicht der Semantik des Ausgangsmodells. Das Zielmodell bietet keine Möglichkeit, das Element wie semantisch gefordert darzustellen oder das Element konnte aus mangelnden Möglichkeiten des Zielmodells nicht übersetzt werden. Daher ist keine Verbesserung der Transformationsregel möglich.

Es war möglich, den Großteil der zu transformierenden Elemente und ihrer Eigenschaften bei der Übersetzung nach Flexsim als erfüllt zu kennzeichnen. Zwei Elemente oder Eigenschaften wurden als erfüllt unter Einschränkungen identifiziert. So erfolgte die Darstellung der `Fork`- und `Join`-Elemente des Ausgangsmodells in Flexsim durch die Elemente `Separator` und `Combiner`. Während die Elemente im Ausgangsmodell mit beliebig vielen Ausgängen ohne Warteschlangenplätze dargestellt werden können, wird in Flexsim für zwei Ausgänge jeweils ein Element mit extra Warteschlangenplatz benötigt. Der Sachverhalt bewirkt eine semantische Änderung im Ablauf des Zielmodells. Einige Elemente wurden als teilweise erfüllt klassifiziert. So war es beispielsweise nicht möglich die Eigenschaften `batch-Size` und `priority` des Elements `SingleProcess` vom Ausgangsmodell nach Flexsim zu übersetzen. Bei der Arbeit der Verifikation und Validierung konnten jedoch gültige Lösungen entwickelt werden. Trotz der fehlenden Eigenschaften erfüllt das Element `SingleProcess` seine restlichen Attribute semantisch korrekt. Da die fehlenden Eigenschaften als Erweiterung der Funktionalität des `SingleProcess` betrachtet werden können, ist die Abweichung als erfüllt unter Einschränkungen zu klassifizieren. Es konnten keine übersetzten Elemente gefunden werden, die als nicht erfüllt zu klassifizieren sind.

Um die Semantik der Übersetzung von SysML nach Flexsim zu testen, wurde eine Sammlung von Testszenarien erstellt, in Flexsim übersetzt sowie anschließend manuell simuliert und bewertet (vgl. Winter, 2012). Dabei kam es zu semantischen Einschränkungen, die teilweise behoben und teilweise nicht behoben werden konnten. Da die Zielmodelle dieser Arbeit Simulatoren sind, die unterschiedliche Metamodelle mit Semantiken und Funktionsumfang besitzen, sind semantische Unterschiede bei verschiedenen Transformationen zu erwarten.

6 Das Modellierungswerkzeug TOPCASED Engineer

Für die Erstellung von Modellen ist ein geeignetes Modellierungswerkzeug notwendig. Dabei ist zu berücksichtigen, dass die Modelle auch von Ingenieuren erstellt werden sollen, die bestimmte Anforderungen an die Gebrauchstauglichkeit eines Werkzeuges stellen. Zudem ist eine nicht-proprietäre Lösung förderlich, um das Modellierungskonzept dieser Dissertation frei nutzen zu können. Daher wurde im Rahmen dieser Arbeit das Modellierungstool TOPCASED *Engineer* entwickelt.

In einem ersten Schritt erfolgte die Betrachtung von Modellierungswerkzeugen für SysML. Anschließend wurden die Anforderungen von Ingenieuren an ein SysML-Modellierungswerkzeug ermittelt. Es wurde entschieden, ein geeignetes Werkzeug zu entwickeln, das im optimalen Fall auf ein Werkzeug unter der *Eclipse Public License* (EPL) oder *Common Public License* (CPL) aufbaut. Die folgenden Ausführungen widmen sich einer Marktuntersuchung von SysML-Werkzeugen, einem *Workshop* zur Ermittlung der *Usability* für SysML-Modellierungswerkzeuge und einer Implementierung eines geeigneten Werkzeuges.

6.1 Marktuntersuchung

Die CPL ist eine Softwarelizenz, die es dem Anwender gewährt, die Software frei zu nutzen, weiter zu bearbeiten oder zu verändern. Die EPL ist eine ähnliche, leicht abgeänderte Version der CPL. Während alle Änderungen am Quellcode von CPL lizensierten Produkten auch wieder mit CPL lizensiert werden müssen, ist dies bei der EPL nicht der Fall. So ist es möglich, Weiterentwicklungen von EPL lizensierten Produkten auch kommerziell zu vermarkten. Die EPL wird beispielsweise für die Eclipse Entwicklungsumgebung und deren *Plugins* verwendet.

Anfang 2009 wurde eine Entscheidung bezüglich eines geeigneten Modellierungswerkzeuges getroffen. Zu diesem Zweck wurde eine Marktuntersuchung von SysML-Modellierungswerkzeugen durchgeführt (vgl. Lißke, 2009). Im Folgenden wird ein Überblick über die betrachteten Werkzeuge gegeben.

Für die Modellierung mit SysML wird ein Werkzeug benötigt, das alle von uns geforderten SysML-Modellierungselemente, ein geeignetes Austauschformat und die Erstellung von großen Modellen unterstützt. Als Austauschformat für SysML-Modelle eignet sich das XML Metadata Interchange (XMI) besonders. Ein großer Vorteil ist, dass XMI auf die wohlstrukturierte, hierarchische Extensible

Markup Language XML aufbaut, durch deren Eigenschaften sich Daten leichter verarbeiten lassen. Auch die Implementierung der aktuellen SysML-Spezifikation ist ein wichtiges Kriterium für ein SysML-Modellierungswerkzeug. Bei der Durchführung der Marktuntersuchung waren die Versionen XMI 2.1 und SysML 1.1 aktuell. Weitere Kriterien waren eine mögliche Erweiterbarkeit des Editors und eine intuitive Bedienung, um den Einstiegsaufwand gering zu halten.

Enterprise Architect wird von der Firma Sparx Systems entwickelt und auch bei der Softwareentwicklung mit UML genutzt. Es unterstützt seit der Version 9.0 SysML 1.2. Sparx Systems bietet außerdem eine Einbindung in Eclipse und *Visual Studio* an. Es werden alle notwendigen Datenaustauschformate von *Enterprise Architect* unterstützt. Mit 250.000 registrierten Nutzern ist es eines der am häufigsten verbreiteten UML-Modellierungswekzeuge (vgl. Sparx Systems, 2013).

Das **SysML-Toolkit** von EmbeddedPlus ist ein *Plug-in* für IBM *Rational's Software Development Platform* (RSDP), welches als Komplettlösung für die Softwareentwicklung entstand. Seit der Version EmbeddedPlus 2.5.1 wird SysML 1.1 unterstützt. Eine Ausgabe der Modelle im XMI-Format ist möglich. Weiterhin zeichnet sich das SysML-Werkzeugkit von EmbeddedPlus durch die Integration in Telelogic *Dynamic Object Oriented Requirements System* (DOORS) aus (vgl. Embedded Star, 2013).

Rhapsody ist eine auf UML 2.1 und SysML 1.0 basierende modellgetriebene Entwicklungsumgebung (MDD) für eingebettete Systeme und Software, die mittlerweile von der Firma IBM vertrieben wird. Rhapsody erlaubt einen Code-zentrierten *Workflow* und die automatische Umsetzung von Code in Modelle. Die Integration mit Eclipse *C/C++ Development Tools* (CDT) ermöglicht Entwicklern das Arbeiten in der Eclipse-Umgebung. (vgl. IBM, 2013).

Artisan Studio Uno ist ein kostenloses Modellierungswerkzeug für SysML und UML. Es ist eine Einzelbenutzerversion von Artisan Studio, das von der Firma Atego entwickelt und vertrieben wird. Artisan Studio selbst bietet alle nötigen Komponenten für die Modellierung von SysML-Modellen. Das Artisan Studio ist ein kostenpflichtiges Programm zur Entwicklung von Software im Team. Das Programm bietet Schnittstellen zu *MATrix LABoratory* (MATLAB), mit dem auch zeitkontinuierliche Modelle erstellt werden können (vgl. Atego, 2013).

MagicDraw ist ein Modellierungswerkzeug für UML und wird von der Firma NoMagic vertrieben. Für MagicDraw existiert ein SysML-*Plugin* welches ab Version 16.0 SysML 1.2 unterstützt. Weiterhin ist es möglich, Modelle im XMI 2.1 Format zu exportieren sowie SysML-Zusicherungen sofort zu validieren. MagicDraw bietet viele Erweiterungen und hat eine komfortable sowie intuitive Benutzeroberfläche (vgl. NoMagic, 2013).

Papyrus 4 UML ist ein *Open-Source*-Modellierungswerkzeug für die Erstellung von UML Modellen und eine Erweiterung für die Eclipse Entwicklungsumgebung. Für Papyrus 4 UML existiert die Erweiterung Papyrus for SysML 1.1 mit der auch SysML-Modelle entwickelt werden können. Eine besondere Eigenschaft von Papyrus ist die erweiterbare Architektur, welche es erlaubt, neue Diagramme und

Code-Generatoren hinzuzufügen. Papyrus ist frei nutzbar und unterliegt den CPL-Bestimmungen. Der Editor war bei der durchgeführten Marktanalyse aber gerade in Bezug auf die SysML-Funktionen noch nicht ausgereift (vgl. Papyrus, 2013).

TOPCASED ist ein *Open-Source*-Modellierungswerkzeug, das ursprünglich für die Entwicklung von kritischen eingebetteten Systemen entwickelt wurde. Auch für TOPCASED gibt es zahlreiche Erweiterungen, so auch TOPCASED-SysML das SysML 1.1 unterstützt und Modelle im XMI 2.1 Format speichert. Der Editor kann als *Plugin* ausgeführt, aber auch in die Eclipse-Entwicklungsumgebung integriert werden. TOPCASED ist ein *Open-Source*-Programm, was den CPL-Bedingungen unterliegt. Somit kann es frei genutzt sowie im Code verändert werden (vgl. TOPCASED, 2013).

Zusammenfassung der Ergebnisse

Bei den kommerziellen Modellierungswerkzeugen konnten besonders MagicDraw und *Enterprise Architect* überzeugen (vgl. Tabelle 6-1). Beide Werkzeuge boten zum Zeitpunkt der Marktuntersuchung den aktuellen SysML- und XMI-Standard. MagicDraw hatte nach ersten Betrachtungen eine besonders intuitive Benutzeroberfläche. Zudem nutzte auch Weilkiens MagicDraw für den Entwurf großer Projekte (vgl. Schulze, 2008). In einer zugrundeliegenden Arbeit wurde evaluiert, dass MagicDraw alle benötigten SysML Funktionalitäten zur Verfügung stellt (vgl. Lißke, 2009). Daher wurde MagicDraw anfänglich als Modellierungswerkzeug für die Modellerstellung dieser Arbeit genutzt.

Ende 2009 wurde ein dieser Arbeit zugrunde liegender *Workshop* veranstaltet, in der die *Usability* von MagicDraw von Informatikern und Ingenieuren getestet wurde. Anhand der Ergebnisse wurde entschieden, ein eigenes Werkzeug zu entwickeln, das spezielle Unterstützung für Ingenieure bereitstellt. Zudem sollte es möglich sein, das Modellierungskonzept vollständig frei zu nutzen. Um den Aufwand einer neuen Implementierung eines Modellierungswerkzeugs zu umgehen, wurde entschieden eines der beiden *Open-Source*-Werkzeuge TOPCASED oder Papyrus zu erweitern. TOPCASED unterstützte damals – im Gegensatz zu Papyrus – den aktuellen SysML- und XMI-Standard, zudem war TOPCASED im Entwicklungsstand ausgereifter. Da TOPCASED der CPL unterliegt, ist es notwendig den veränderten Code auch unter die CPL zu erstellen. Zusätzlich erstellte Module, die aus dem Produkt heraus aufgerufen werden (wie ein Transformator oder Simulator), könnten bei Bedarf jedoch kommerziell vermarktet werden.

Tabelle 6-1: Zusammenfassung der Marktuntersuchung SysML Modellierungstools

Attribut	SysML 1.0	SysML 1.1	XMI 2.0	XMI 2.1	Lizenz	Eclipse basiert
Enterprise Architekt	X	X	X	X	kom-merziel	X
SysML Toolkit	X	X	?	?	kom-merziel	X
Telelogic	X		X	X	kom-merziel	X
Artisan Studio	X	?	X		frei nutzbar	
MagicDraw	X	X	X	X	kom-merziell	
Papyrus	X		X		open source	X
Topcased	X	X	X	X	open source	X
Rhapsody	X		X	X	kom-merziell	X

6.2 Spezielle Anforderungen an Modellierungswerkzeuge für Ingenieure

Um die praktische Anwendbarkeit von SysML zu testen, wurde ein größeres Projekt aus dem Bereich der Montage mit verschiedenen Modellierungssprachen und Werkzeugen getestet (vgl. Gsuck, 2009). Dabei wurde unter anderem die Modellierungsqualität von SysML mit MagicDraw betrachtet. Als positiv wurden die vielfältigen Modellierungsmöglichkeiten von SysML evaluiert. Im Vergleich zu anderen Modellierungssprachen wie beispielsweise MoogoNG konnte das komplette Szenario ohne Einschränkungen modelliert werden(vgl. Gsuck, 2009, S.11ff.). Die Modellierung komplexer Systeme zeigte sich bei geübten SysML-Nutzern als komfortabel und schnell. Jedoch wurde der Einarbeitungsaufwand als nicht unerheblich bewertet, da die Modellierung mit SysML nicht trivial ist. Obwohl die kommerziellen Werkzeuge übersichtlich und komfortabel waren, zeigte sich die *Usability* gerade bei der Einarbeitung als nicht optimal (Gsuck, 2009, S48ff.).

Nach den Ergebnissen von Gsuck wurde ein *Workshop* durchgeführt, um die *Usability* von SysML vertiefend zu evaluieren und Verbesserungsmöglichkeiten zu finden. Am *Workshop* nahmen achtzehn Personen aus den Bereichen Informatik und Ingenieurwesen teil. Die Testpersonen sollten zwei Szenarien mit dem Modellierungswerkzeug MagicDraw modellieren. Nach der Modellierung eines Szenarios wurde die Modellierung mit SysML und MagicDraw anhand eines Evaluationsbogens bewertet. Die Testpersonen wurden nach ihrem Erfahrungsschatz im Umgang mit UML und SysML eingeteilt. Es wurde bewertet, in welcher Zeit die Personen das Szenario mit welcher Qualität bewältigten. Zudem konnten die Testpersonen Verbesserungsvorschläge für das Modellierungswerkzeug angeben, um die Modellierung der Modelle mit SysML zu vereinfachen. Die Evaluationsbögen sind dem Anhang beigefügt.

In den Ergebnissen zeigte sich, dass die Modellierung der Systeme für erfahrene Personen deutlich einfacher war. Gerade die Ingenieure hatten erhebliche Probleme bei der Einarbeitung. Dies kann daran liegen, dass sie im Gegensatz zu den Informatikern weder mit SysML noch mit UML Erfahrungen hatten. Grundlegend konnten aus den Verbesserungsvorschlägen, zwei prinzipielle Probleme beim Umgang mit SysML ermittelt werden:

- SysML ist mächtig und die Anzahl seiner Modellierungsmöglichkeiten ist groß.
- Die Modellierung mit SysML ist sehr abstrakt (was SysML unter anderem mächtig macht).

Aus den genannten Gründen wurde beschlossen, ein SysML-Modellierungswerkzeug zu entwerfen, welches an die Anforderungen von Ingenieuren bei der Modellierung diskreter Systeme angepasst wird. Abbildung 6-1 zeigt, dass SysML weit mehr Modellierungsmöglichkeiten bereitstellt, als für das Modellieren von diskreten Prozessen benötigt werden. Hier ist es möglich, den Ingenieur durch ein angepasstes Werkzeug in seiner Auswahl zu unterstützen.

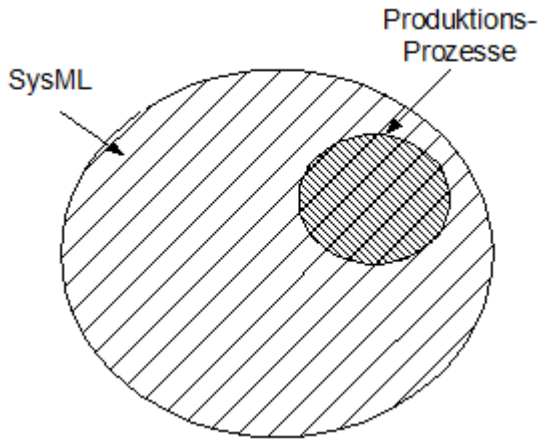


Abbildung 6-1: Modellierungsmöglichkeiten von Produktionsprozessen in SysML

Wie im zweiten Kapitel ausgeführt, kann ein SysML-Block alles Mögliche darstellen, wie beispielsweise eine Ressource, einen Prozess, ein chemisches Element oder eine physikalische Einheit. Dabei handelt es sich um eine sehr abstrakte Denkweise, welche bei komplexer Modellierung zu Verwirrungen führen kann. Gerade Ingenieure, deren bisherige Werkzeuge eine Menge fest definierter, anwendungsspezifischer Elemente zur Verfügung stellten, kann dies irritieren. SysML bietet wie im Kapitel 2.3 gezeigt die Möglichkeit, so genannte Stereotype zu nutzen, welche ein definiertes Element mit Eigenschaften darstellen. Ein geeignetes Modellierungswerkzeug kann mit Stereotypen, eine fest definierte Menge von Elementen zur Verfügung stellen. Um die Modellierung aller möglichen Szenarien im Untersuchungsgegenstand abzudecken, muss eine große Zahl an Stereotypen und derer Eigenschaften zur Verfügung gestellt werden. Um trotzdem die Übersicht zu wahren, kann dem Ingenieur vor der eigentlichen Modellierung eine bearbeitbare Vorauswahl an Stereotypen und Eigenschaften zur Ver-

fügung gestellt werden. Dies ist in MagicDraw beispielsweise nicht möglich. Daher wurde ein Modellierungswerkzeug entwickelt, welches auf dem *Open-Source*-Projekt TOPCASED aufbaut. Bei der Entwicklung des Werkzeuges, wurden die Erkenntnisse des *Workshops* genutzt, um die Gebrauchstauglichkeit speziell für Ingenieure anzupassen.

6.3 Praktische Betrachtungen des Modellierungswerkzeuges TOPCASED Engineer

Das *Toolkit in Open Source for Critical Applications & Systems Development* (TOPCASED) ist ein Modellierungswerkzeug, das für die Umsetzung von Projekten aus der Luft- und Raumfahrttechnik sowie im Fahrzeugbau entwickelt wurde. Zur Umsetzung wurden in TOPCASED die Modellierungssprachen UML, SysML, SAM und ADDL implementiert. *Structured Analysis Modeling* (SAM) ist eine Modellierungssprache für die Transformation und Verifizierung von funktional strukturierten Analysen. Die *Architecture Analysis & Design Language* (AADL) ist eine standardisierte Beschreibungssprache für den Kraftfahrzeugbau und Flugzeugbau. Für diese Arbeit ist nur der TOPCASED Teilbereich SysML von Bedeutung. Die Entwickler haben die vollständigen Metamodelle der SysML und UML nach den Vorgaben der OMG erstellt. Dabei hielten sie sich strikt an die Vorgaben der OMG und ließen keine eigenen Erweiterungen einfließen (Lißke, 2009, S. 21). Durch den Anwendungsbereich, der sich auf mehrere Modellierungssprachen erstreckt, besteht TOPCASED aus 141 verschiedenen Softwaremodulen. Für die Darstellung von SysML benötigt TOPCASED lediglich 21 Module. Für eine erleichterte Bearbeitbarkeit wurde TOPCASED in dieser Arbeit auf die für SysML nötigen Module verringert (Lißke, 2009, S. 23).

Das TOPCASED *Open-Source*-Projekt ist wie die meisten *Open-Source*-Anwendungen nicht fehlerfrei. Die Anforderung dieser Arbeit an ein geeignetes Modellierungswerkzeug ist eine möglichst benutzerfreundliche Umsetzung. Da die Anwender des Modellierungskonzepts hauptsächlich Ingenieure sind, muss das Modellierungswerkzeug den Ingenieur in seinen Anforderungen unterstützen. Um TOPCASED an die Anforderungen dieser Arbeit anzupassen, war es notwendig TOPCASED:

1. von Fehlern der Entwicklung zu bereinigen,
2. allgemein in seiner *Usability* zu verbessern,
3. und den speziellen Anforderungen von Ingenieuren anzupassen.

Um diese Anforderungen umzusetzen wurden vier studentische Arbeiten durchgeführt (vgl. Scharfe, 2011; vgl. Sharapov, 2010; vgl. Frank, 2010; vgl. Moss, 2012). In den folgenden Ausführungen werden die Veränderungen an TOPCASED im Überblick zusammengefasst. In den aufgeführten Arbeiten ist es möglich, die Veränderungen detailliert und mit ihren Implementierungsdetails einzusehen.

6.4 Abänderungen des TOPCASED Engineers

Abbildung 6-2 zeigt den originalen Aufbau der TOPCASED Modellierungs-Perspektive, in der Abbildung 6-3 ist der für diese Arbeit abgeänderte Aufbau zu sehen:

Der TOPCASED **Package Explorer** (1) bietet dem Nutzer die Möglichkeit, zwischen verschiedenen Diagrammen und Projekten zu wechseln. Zugleich konnten durch den *Package Explorer* im Editor (3) Diagramme aus verschiedenen Projekten gewählt werden. In der praktischen Arbeit und dem *Workshop* führte der *Package Explorer* zu zahlreichen Missverständnissen, da die Anwender Schwierigkeiten hatten, zwischen Projekten und ihren Diagrammen zu unterscheiden. Die kommerziellen Werkzeuge wie MagicDraw arbeiten daher standardmäßig in einem Projekt und bieten im Editor nur Karteikarten von verschiedenen Diagrammen des Projektes an. Dieser intuitiven Lösung wurde sich angeschlossen, so dass der *Package Explorer* entfernt wurde und der Editor nur noch Diagramme eines Projektes zur Auswahl stellt. Wenn ein anderes Projekt bearbeitet werden soll, wird der ganze Arbeitsplatz auf das neue Projekt umgestellt. Dies stellt eine konzeptionelle Änderung von TOPCASED dar.

In der unteren TOPCASED **Toolbar** (2) kann der Anwender die Elemente des jeweiligen SysML-Diagrammtyps zum Modellieren eines Szenarios auswählen. Im *Workshop* wurde evaluiert, dass gerade die Ingenieure durch die große Anzahl an SysML-Elementen verwirrt werden. Das Modellierungskonzept dieser Arbeit benötigt nur einen Teil der Elemente (vgl. Abbildung 6-3). Die Toolbar wurde so angepasst, dass dem Nutzer nur Elemente die zum Umsetzen des Modellierungskonzeptes benötigt werden, zur Verfügung stehen. Beim Blockdefinitionsdiagramm werden dem Anwender zusätzlich vorausgewählte Stereotype zur Verfügung gestellt.

Im TOPCASED **Editor** (3) kann der Anwender die Szenarien durch die Elemente der Toolbar modellieren. Die verschiedenen Diagramme sind mittels der TOPCASED *Outline* (4) oder durch Karteikarten im Editor auswählbar. Die Darstellung der einzelnen Elemente des Blockdefinitionsdiagrammes und des Aktivitätsdiagrammes wurde zudem für eine bessere Übersicht bearbeitet, worauf in folgenden Ausführungen noch detailliert eingegangen wird.

Die TOPCASED **Outline** (4) bietet dem Anwender eine Übersicht des ausgewählten Diagramms und eine Liste vom Inhalt des SysML-Modells als Baumstruktur. Der Anwender kann hier neue Diagramme und Elemente über das *Context Menu* hinzufügen, entfernen und über *Drag-and-Drop* in den Editor einfügen. Im Gegensatz zum originalen TOPCASED wurde die *Outline* an der Stelle des *Package Explorers* angeordnet, wodurch der Editor mehr Platz erhält, um die Diagramme darzustellen.

Wenn der Anwender im *Editor* oder der *Outline* die rechte Maustaste betätigt, wird das TOPCASED **Context Menu** (5) aufgerufen. Hier wurden einige Erweiterungen für eine bessere *Usability* vorgenommen auf die in folgenden Ausführungen eingegangen wird.

Der TOPCASED **Properties Tab** (6) gibt dem Anwender die Möglichkeit, im Editor markierte Elemente zu bearbeiten. Hier wurden Änderungen bei der Anordnung der Menüs und bei den Eingabemöglichkeiten vorgenommen, die folgend genauer betrachtet werden.

In der TOPCASED **Menubar** (8) kann der Anwender verschiedene Optionen bearbeiten, in der oberen **Toolbar** (7) können wichtige Funktionen aufgerufen werden. Hier wurde die Toolbar den benötigten Funktionsumfang angepasst.

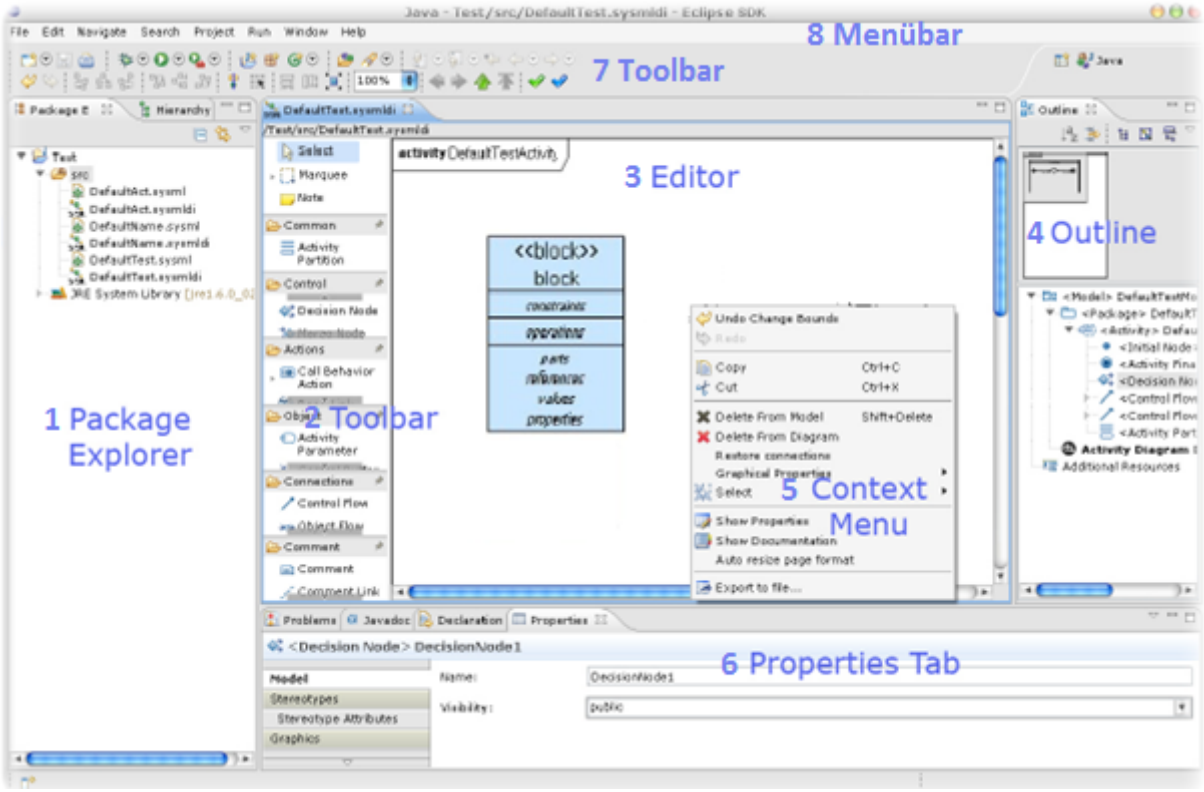


Abbildung 6-2: Aufbau der ursprünglichen TOPCASED-Oberfläche

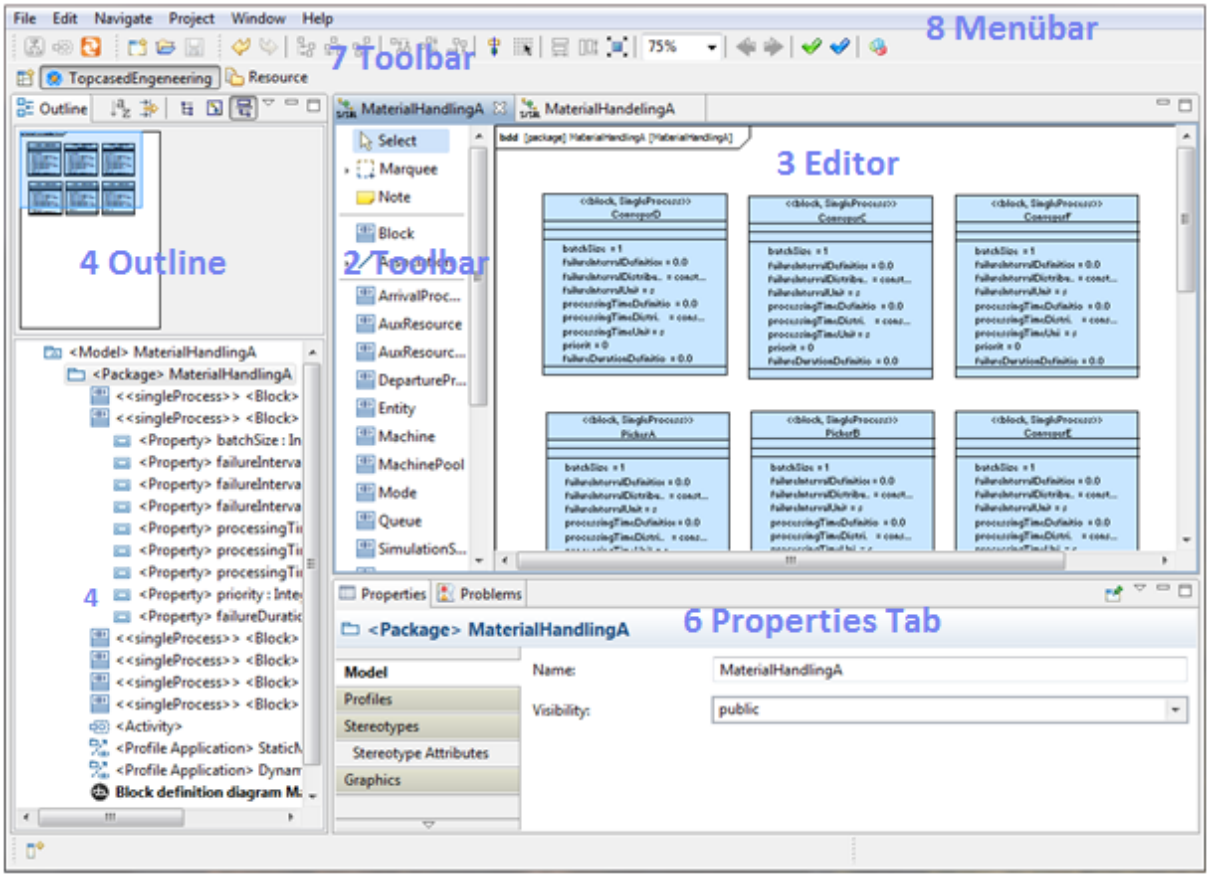


Abbildung 6-3: Aufbau der Oberfläche des TOPCASED *Engineer*

Die Erstellung eines neuen Projektes wurde zur Verbesserung der *Usability* grundlegend verändert. Im letzten Schritt beim Erstellen eines neuen Projektes bekommt der Anwender ein Fenster für Voreinstellungen der für das Projekt relevanten Stereotypen. Hier ist es möglich, zu wählen, welche Stereotypen mit welchen Eigenschaften für das Projekt in welcher Farbkombination genutzt werden. Diese erscheinen dann beim Bearbeiten des Projektes in der Toolbar. Während der Arbeiten am Modell kann die Auswahl durch einen in der Toolbar neu eingebundenen Button bearbeitet werden (vgl. Abbildung 6-4).

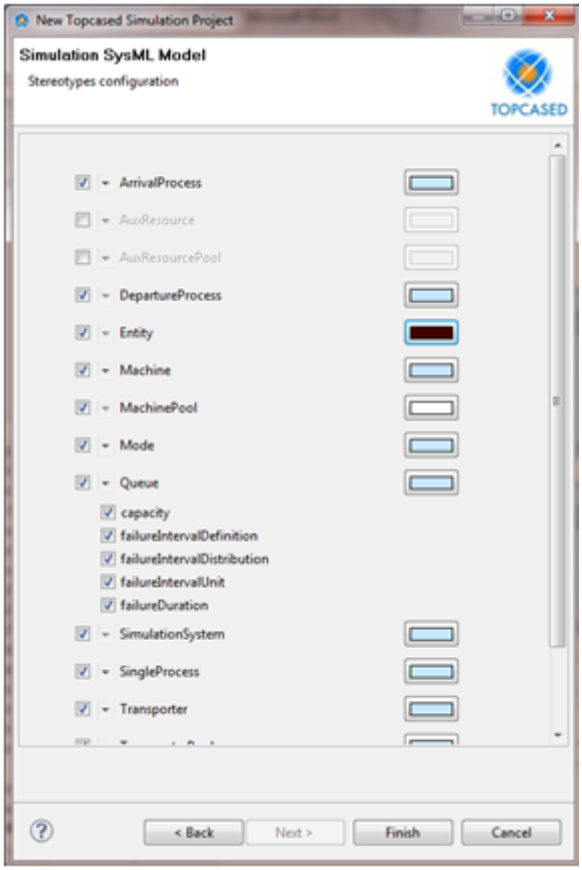


Abbildung 6-4: Auswahl der Stereotypen im TOPCASED *Engineers*

Die Stereotypen-Darstellung im Editor wurde für einen besseren Überblick angepasst. Während im Original TOPCASED ein Stereotyp mit den Schlüsselwörtern eines SysML-Blocks angezeigt wurde (vgl. Abbildung 6-5 links), werden jetzt die definierten Attribute eines Stereotyps und deren zugeordnete Werte angezeigt (vgl. Abbildung 6-5 rechts). Diese Änderung ist mit der Darstellung kommerzieller Editoren konform und für die übersichtliche Darstellung von Projekten notwendig. Auch bei der Darstellung von Aktivitätsdiagrammen wurden Änderungen eingepflegt. So war es beim Original TOPCASED nur möglich `ActionNodes` durch `CallBehaviorActions` darzustellen. Weiterhin lassen sich `ActivityPatitions` jetzt durch eine einfache Funktion automatisch ausrichten. Sollen in TOPCASED mehrere Elemente verbunden werden, ist die Verbindung bei jedem Paar neu aus der

Toolbar auszuwählen. Im TOPCASED *Engineer* lassen sich die Elemente nun durch Betätigen der Shift-Taste seriell verbinden.

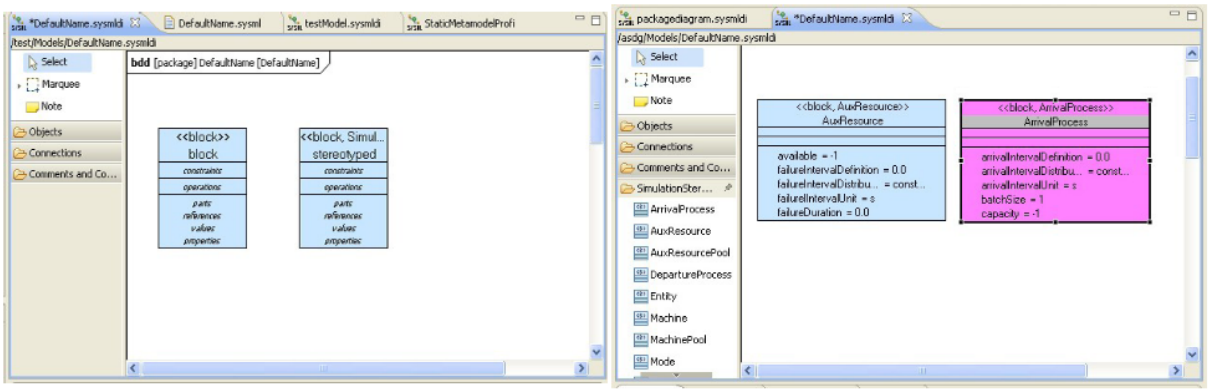


Abbildung 6-5: Darstellung von Stereotypen beim TOPCASED *Engineers*

Wird ein Element im Editor angewählt, öffnet sich das Properties Tab. Hier kann der Anwender die verschiedenen Eigenschaften eines Elements definieren. Um die Nutzerfreundlichkeit des TOPCASED *Engineers* zu verbessern, wurde die Menüführung überarbeitet. Zudem wurde die Werteeingabe bei Enumerationen so abgeändert, dass dem Nutzer eine Liste der möglichen Werte vorgegeben wird (vgl. Abbildung 6-6).

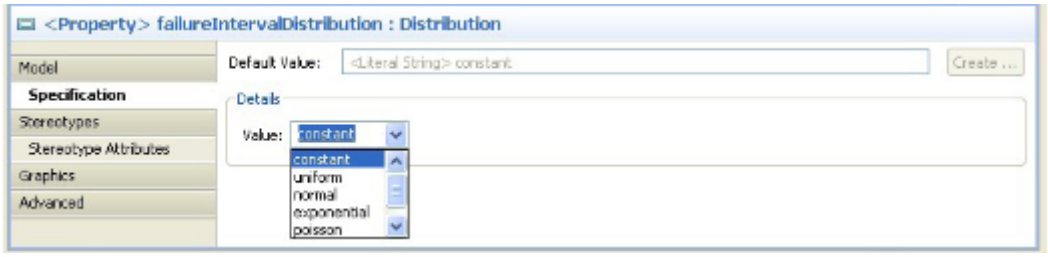


Abbildung 6-6: Änderungen des Properties Tabs

Die TOPCASED Toolbar enthält eine Vielzahl an Elementen, die für die Nutzung des Programmes im Kontext dieser Arbeit nicht benötigt werden. Abbildung 6-7 zeigt auf der linken, rot eingerahmten Seite alle nicht benötigten Elemente, die im TOPCASED *Engineer* entfernt wurden. Auf der rechten, rot eingerahmten Seite des TOPCASED *Engineers* sind zusätzlich eingefügte Elemente, für das einfache Zufügen von Diagrammen, das Starten des Übersetzungswerkzeuges und das Abändern von Stereotypen aufgeführt.

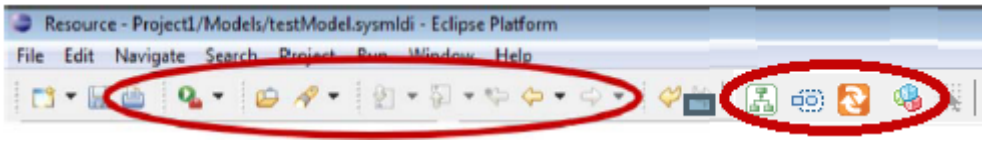


Abbildung 6-7: Änderungen des Properties Tabs

Der Kontextbaum in der Outline wurde so angepasst, dass ein Package durch den Aufruf des Context Menüs um alle notwendigen Diagramme erweitert werden kann (vgl. Abbildung 6-8).

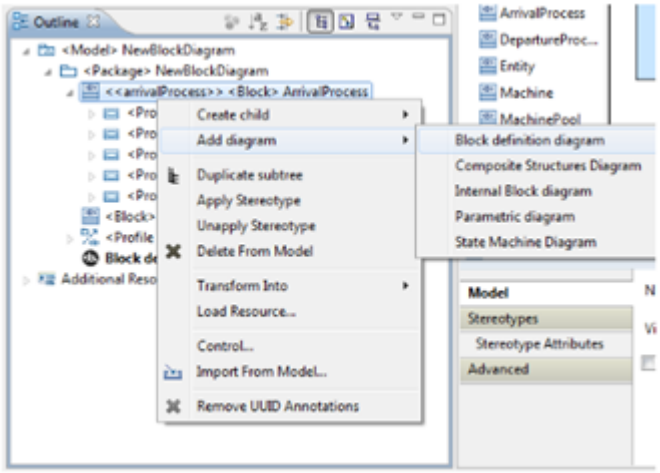


Abbildung 6-8: Erweiterung des Kontextmenüs der Outline

Im ursprünglichen TOPCASED unterliegen Diagramme der Beschränkung einer Maximalgröße. Um diese Restriktion zu umgehen, wurde das unbegrenzte Seitenformat `Modifiable` eingeführt. Werden in diesem Seitenformat Diagramme bearbeitet, ist es möglich, durch das Einfügen von Elementen im Editor über dem Diagrammrahmen, das Diagramm durch *Drag-and-Drop* zu vergrößern.

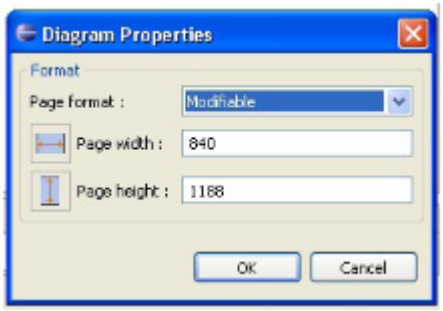


Abbildung 6-9: Unbeschränkte Diagrammgröße durch den Datentyp `Modifiable`

7 Ein Simulationswerkzeug basierend auf JAMES II

Neben der Umsetzung des Modellierungskonzeptes für kommerzielle Simulatoren ist es eine interessante Idee, eine Übersetzung für ein nicht-proprietäres Simulationswerkzeug zu implementieren, um so eine ganzheitliche nicht proprietäre Lösung zur Verfügung zu stellen. In den durchgeführten Marktuntersuchungen konnte jedoch kein geeignetes nicht proprietäres Werkzeug gefunden werden, welches das Modellierungskonzept der Dissertation umsetzen kann. Daher wurde die Entscheidung getroffen, einen auf dem Modellierungskonzept dieser Arbeit eigenentwickelten Simulator zu implementieren. Um den Aufwand für die Implementierung möglichst gering zu halten, wurde ein geeignetes *Framework* gesucht. Die Universität Rostock bietet das ausgereifte und freie *Framework* JAMES II zum Erstellen von Simulatoren (vgl. JAMES II, 2013). Auf der Basis von JAMES II und den Ergebnissen dieser Arbeit wurde die Neuimplementierung eines Simulators unternommen.

Der erste Abschnitt des Kapitels zeigt die Systemarchitektur des Simulationssystems und wie die einzelnen Komponenten miteinander agieren. Anschließend wird auf JAMES II und die vom *Framework* zur Verfügung gestellten Komponenten eingegangen. Der dritte und vierte Abschnitt des Kapitels zeigen die Umsetzung und Integration des Simulationsalgorithmus und des Modells in JAMES II.

7.1 Systemarchitektur

Abbildung 7-1 zeigt die Architektur des Simulationssystems. In einem ersten Schritt werden die Modelle mit dem TOPCASED *Engineer* erstellt. Anschließend überträgt der *Parser* die Informationen in das interne Datenmodell. Der auf JAMES II basierende Simulator verwendet die Informationen des internen Modells direkt als Modellbeschreibung und simuliert das Szenario. Abschließend überträgt der *Parser* die Ergebnisse der Simulation aus dem internen Modell nach TOPCASED. Der Anwender kann die Ergebnisse dann in TOPCASED einsehen. TOPCASED *Engineer*, der *Parser* in beide Richtungen, das Modellierungskonzept sowie das interne Modell sind schon implementiert, was den Entwicklungsaufwand einschränkt. Somit muss nur der Simulator implementiert werden. Das *Framework* JAMES II erleichtert die Aufgabe zusätzlich. Die Architektur des Simulators basiert auf der von eventgesteuerten Simulatoren (vgl. Kapitel 3.1).

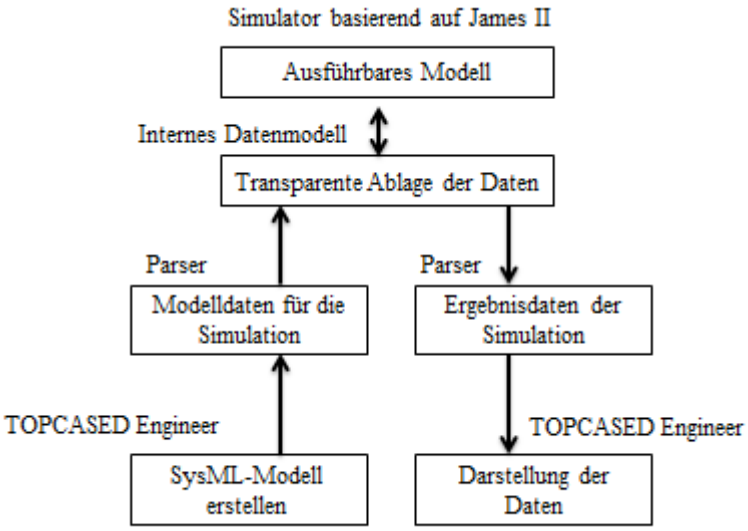


Abbildung 7-1: Systemarchitektur des Simulationswerkzeuges

Abbildung 7-2 zeigt die Zusammenhänge der einzelnen Komponenten des Simulationssystems. TOPCASED *Engineer* basiert auf der standardisierten Modellierungssprache SysML. Zudem basiert TOPCASD *Engineer* auf einem Profil, das vom Modellierungskonzept dieser Arbeit abgeleitet wird. Das Simulationsmodell unterliegt somit den Restriktionen des Modellierungskonzeptes. Ausgeführt wird es mit dem implementierten Simulator, der auf dem Modellierungskonzept der Arbeit und JAMES II aufbaut.

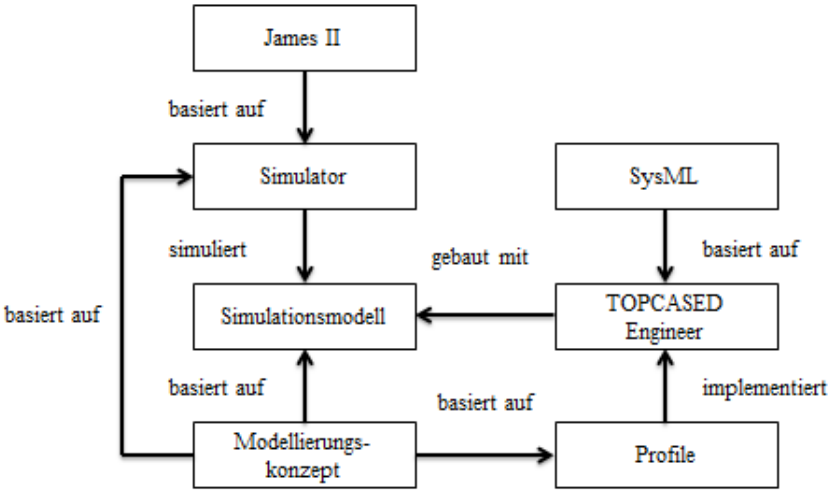


Abbildung 7-2: Zusammenhang der einzelnen Komponenten des Simulationssystems

7.2 JAMES II

JAMES II ist ein *Framework*, das *Plugin*-basiert verschiedene Datenstrukturen und Formalismen zur Verfügung stellt (vgl. Himmelspach/Uhrmacher, 2009). *Plugin*-basiert bedeutet in diesem Fall, dass JAMES II, ähnlich wie die Eclipse-Umgebung, einen festen Kern besitzt, der lediglich das Laden und Einbinden von *Plugins* zur Verfügung stellt. Jede zusätzliche Funktionalität, die im *Framework* bereits besteht, wurde von den JAMES Entwicklern als *Plugin* ergänzt. Die Hauptteile des *Frameworks* sind nach Himmelspach und Uhrmacher:

- User Interface: JAMES II hat ein eingebautes *User Interface* für die Modellierung experimenteller Einstellungen, die Analyse von Ergebnissen und die Visualisierung des Simulationslaufes. Bisher wird in dieser Arbeit kein Gebrauch vom *User Interface* gemacht, da die Eingaben und Ausgaben über den TOPCASED *Engineer* erfolgen.

- Experiment: JAMES II enthält verschiedene Methoden, um Experimente in JAMES II zu definieren. Mithilfe des Experiments ist es möglich, mehrere Simulationsläufe und ihre Parameter zu definieren. Bisher wird das Paket noch nicht in dieser Arbeit genutzt. In zukünftigen Betrachtungen ist zu testen, in wie weit das JAMES Paket `Experiment` mit dem Experimentellen-Modell dieser Arbeit verbunden werden kann.

- Database: In einer Simulation kommt es oft zur Erzeugung großer Datensätze. Das Paket `Database` stellt ein *Interface* für die Anbindung von Datenbanken, macht aber keine Vorgaben, wie die Daten gespeichert werden sollen. Bei einem fortgeschrittenen Arbeitsstand ist dieses Paket eine angenehme Hilfestellung.

- Model: Dies ist das Basis *Interface* von JAMES II, für die Erstellung von Modellen. Alle neuen Modellierungs-Formalismen, die mit JAMES II simuliert werden sollen, müssen von `Model` erben. So nutzt auch diese Arbeit den Vorteil der wohldefinierten Datenstrukturen des Pakets `Model`.

- Simulator: Das Paket `Simulator` wird benötigt, um ein Modell zu simulieren. Das Paket ist der Kern von JAMES II, er enthält keinen festgelegten Simulationsalgorithmus, bietet aber viele Möglichkeiten, neue Simulationsalgorithmen hinzuzufügen. In dieser Arbeit wurde ein Simulationsalgorithmus für die diskrete, eventgesteuerte Simulation eingeführt.

- Simulation: Im Paket Simulation wird die Ausführung eines Modells mit einem speziellen Simulator definiert. Es ist möglich verteilte und einfache Simulationen zu definieren. Bei verteilten Simulationen werden Mechanismen bereitgestellt, mit denen Partitionierungen und Verteilungen definiert werden. Bisher arbeitet der Simulator dieser Arbeit nicht verteilt. Es ist aber gut vorstellbar, eine spätere Umstellung durch das Arbeitspaket zu erleichtern.

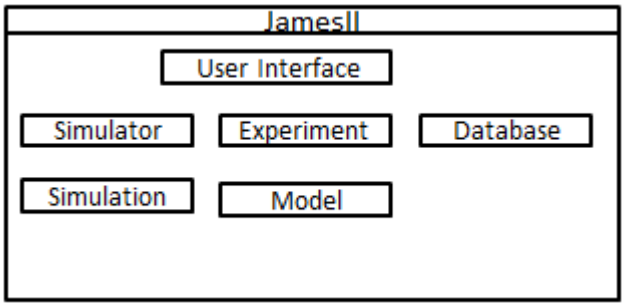


Abbildung 7-3: Arbeitspakete JAMES II

Neben den funktionalen Komponenten hat JAMES II eine Registrierung, in der alle benötigten *Plugins* dynamisch zum Start geladen werden. Die Registrierung enthält Methoden, um *Plugins* während der Laufzeit zu listen und hinzuzufügen. Um seine Funktionalität zu erweitern, enthält JAMES II ein *Plugin*-Konzept. Jedes *Plugin* ist einem *Plugin*-Typen zuzuordnen. Neue *Plugins* werden in einem XML-File deklariert. Die Implementierung wird in Java durchgeführt. Ein anderes wichtiges Konzept ist die Trennung des Modells vom Simulationsalgorithmus. Der Simulationsalgorithmus kann ein Modell ausführen. Das Modell hingegen hat keinen Zugang zum Simulationsalgorithmus. So ist es möglich, verschiedene Simulationsalgorithmen auf ein Modell anzuwenden. Um es zu ermöglichen, SysML-Modelle des Modellierungskonzeptes mit JAMES II zu simulieren, müssen zwei Schritte ausgeführt werden:

1. Das interne Modell im Modell-*Interface* von JAMES II zu integrieren und mit semantischen Details zur Ausführung zu erweitern.
2. Einen Simulationsalgorithmus zu erstellen. Alle Simulationsalgorithmen müssen das *Interface* `IProcess` implementieren. Die wichtigste Methode zum Ausführen eines Simulationsalgorithmus ist die Methode `nextStep()`. Sie legt den nächsten Schritt fest. Bei diskreten eventgesteuerten Simulatoren ist der nächste Schritt der zeitlich nächste Schritt.

Der fertige Simulator kann letztendlich als *Plugin* in TOPCASED *Engineer* eingebunden werden.

7.3 Umsetzung des Simulationsalgorithmus in JAMES II

Für die Modellierung des Simulationskonzeptes wurde ein diskreter, eventbasierter Simulationsalgorithmus implementiert. Die *Events* sind daher das zentrale Element der Simulation. *Events* werden an definierten Zeitpunkten gestartet. Grundlegend verwaltet der Simulator eine Eventtabelle und führt bei jedem Schritt der Simulation die folgenden Abläufe aus (vgl. Abbildung 7-4):

1. Alle *Events* des folgenden Zeitpunktes aus der Eventtabelle holen.
2. Simulationszeit der in den *Events* hinterlegten Zeit anpassen.
3. *Events* an deren Empfänger senden, um sie abzuarbeiten.

4. Folgeevents sammeln und in die Eventtabelle eintragen.

5. Wenn die Eventtabelle nicht leer ist, springe zurück zu Punkt 1, ansonsten können die Ergebnisse aufbereitet und die Simulation beendet werden.

Abbildung 7-4: Arbeitsschritte des Simulationsalgorithmus

Dieser einfache Ansatz wurde in der konkreten Umsetzung des implementierten Simulationsalgorithmus erweitert, so dass drei verschiedene Eventtabellen verwaltet werden. Es wird jeweils eine extra Liste für *Events* aus Routineentscheidungen des dynamischen Modells, *Events* für Fehler und sonstige Ereignisse angelegt. In die Eventliste für Routingentscheidungen kommen Ereignisse, die im Verhalten (den Aktivitätsdiagramm) durch die wandernden *Entities* (Token) ausgelöst werden. So wird beim Austritt eines Prozesses ein Token erzeugt, der beispielsweise durch ein `ForkNode` dupliziert werden kann. Tritt ein Token in eine Aktion (`SingleProcess, Queue` oder `DepartureProcess`) ein, wird das *Event* aus der Ereignisliste für Routingentscheidungen gelöscht und ein neues in der Liste für sonstige *Events* angelegt. Mit diesem ist es dann beispielsweise möglich, einen Prozess aufzurufen, der wiederum neue *Events* für das Binden von Ressourcen, das Starten des Prozesses oder sonstige *Events* erzeugt. Letztere werden wieder in der Eventliste für sonstige Ereignisse abgelegt. Zudem gibt es eine Liste für Fehler. Fehlerevents stehen in Konflikt mit den herkömmlichen Simulationsevents. Das Auftreten eines Fehlers einer Ressource zum Beispiel beeinflusst auch den Prozess, an dem die jeweilige Ressource im gleichen Moment arbeitet. Beim Start der Simulation wird für jedes Element, das Fehler enthalten kann, nach den angegebenen Verteilungen der nächste auftretende Fehler berechnet und in die Eventliste für Fehler eingetragen. Wird ein Fehlerevent erreicht, werden die *Events* der Standard-Eventliste korrigiert (Prozessverzögerungen und andere Einflüsse), zudem wird der nächste auftreten-

de Fehler berechnet und in die Liste eingetragen. Im Gegensatz zur Eventliste für das Routing und zur sonstigen Eventliste kann die Simulation auch terminieren, wenn noch *Events* in der Fehlereventliste eingetragen sind (vgl. Abbildung 7-5).

Während die Fehlerereignisliste und die Eventliste für Routingentscheidungen mit einfachen *Hashtables* realisiert werden, wird für die Liste sonstiger *Events* eine spezielle Datenstruktur verwendet. In JAMES II sind Datenstrukturen vorgefertigt, die eine Eventtabelle um eine zusätzliche Tabelle für zeitnahe Events erweitern und die Synchronisation der beiden Tabellen übernehmen. Während die größere Menge an zeitfernen *Events* in der originalen Eventtabelle bleibt, werden die zeitnahen *Events* in eine neue Tabelle sortiert. So ist es möglich, die üblicherweise gehäuften Zugriffe der zeitnahen *Events* – wie beispielsweise durch Vergleichsoperatoren – in kürzerer Zeit durchzuführen. Die Datenstruktur für die Sonderbehandlung der beiden Eventlisten wird durch die komplexe JAMES-II-Datenstruktur `IEventQueue` aus zwei Teilstrukturen zusammengesetzt. Während die zeitnahen *Events* in der Datenstruktur `Head` behandelt werden, werden die zeitentfernten in der Datenstruktur `Tail` behandelt. So ist es möglich, die Vorteile der Java Datenstrukturen `LinkedList` und `TreeSet` zu koppeln. Von außen ist der Zugriff auf die beiden Datenstrukturen homogen. Die Strukturen wurden für den Zugriff, das Einfügen und das Entfernen von *Events* optimiert. Zudem kann der Entwickler selbst `Timetables` erweitern oder implementieren.

JAMES II bietet weitere komplexere Simulationsansätze, die implementiert werden können. Für den vorliegenden Aufgabenkontext wurde die Implementierung eines einfachen `RunnableProcessor` gewählt. Eine Weiterentwicklung des hier gewählten Simulationsansatzes wäre beispielsweise eine verteilte Simulation.

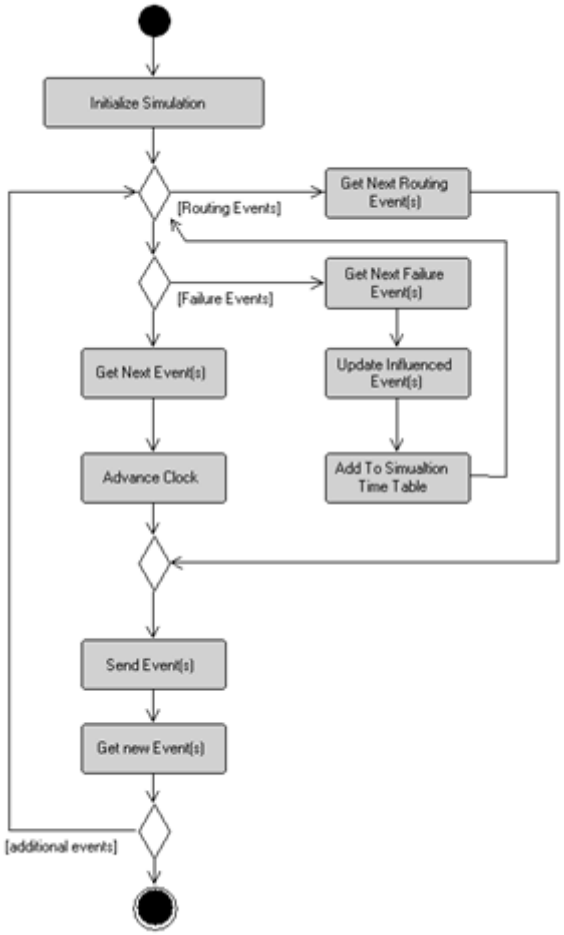


Abbildung 7-5: Arbeitsschritte des erweiterten Simulationsalgorithmus

7.4 Integration des Modells in JAMES II

Um Simulationen auf den Basismodellen durchzuführen, bedarf es innerhalb von JAMES II einer Klassenrepräsentation eines ausführbaren Modells. Ein Modell muss, um in JAMES II als ausführbar deklariert zu sein, das *Interface* `IModel` implementieren. Dieses *Interface* stellt Methoden zur Initialisierung und zur Bereinigung nach der Simulation bereit. Das interne Modell selbst ist in einer Klassenstruktur abgelegt, dessen Wurzelelement eine Instanz von `SimulationSystem` ist und `IModel` implementiert. Für die Simulationsinitialisierung in JAMES II werden in den einzelnen Elementen Zustände, eventuelle Warteschlangen und statistischen Verteilungen initialisiert.

Bei anderen Strukturen war es ebenfalls notwendig, Anpassungen vorzunehmen. Ein einzelnes **Entity**-Element im internen Modell, das mehrere *Units* als *Lot* besitzt, ist im internen Modell ein Element mit einem Attribut. Für die Simulation hingegen ist ein einzelnes Element zur Berechnung nicht ausreichend. In den Ankunftsprozessen werden Intervall-Objekte erzeugt, die dann tatsächlich durch die Prozesskette wandern. Dabei ist zwischen `FlowUnit`, also einer einzelnen Einheit, und `FlowLot` zu

unterscheiden. Ein *Lot* beinhaltet eine Reihe von *Units*. Die verschiedenen *Units* eines *Lots* werden in einer dem *Lot* zugeordneten Tabelle gespeichert. Zur Unterscheidung von der *Entity*-Klasse wurden `FlowItems` definiert, die die bewegten Objekte repräsentieren.

Auch **Ressourcen** sind auf die Anforderungen der Simulation vorzubereiten. Es wird intern genau ein Ressourcenpool erstellt, der alle vorhandenen Ressourcen verwaltet. Die Ressourcen selbst werden entsprechend ihrer im statischen Modell definierten Anzahl repliziert, um als separate Instanzen zur Verfügung zu stehen.

Der **Monitor** dient der Überwachung und Synchronisation der Simulation. Er ist als *Observer*-Entwurfsmuster implementiert und registriert sich entsprechend bei jedem einzelnen Modellelement. Bei jedem Zustandswechsel der Elemente werden die *Observer* – und somit der Monitor – über den Zustandswechsel informiert. Letzterer aktualisiert daraufhin seine vorhandenen Informationen über das Element und prüft, ob das Element bei dem Zustandswechsel neue Events produziert hat. Die Events werden, wenn vorhanden, vom Monitor zum Simulator weitergeleitet. Generell ist der Monitor die einzige Schnittstelle zwischen den Modellelementen und dem Kontroller, den Modellelementen und dem Simulator, sowie dem Kontroller und dem Simulator.

Sollen in einem Prozess `FlowItems` weitergeleitet werden, erzeugt das Element ein entsprechendes Anfrage-*Event*, und der Monitor fügt das *Event* in die Eventtabellen des Simulators ein. Im nächsten Schritt wird das *Event* an den **Kontroller** übermittelt. Sind nun Anfragen für mehrere Elemente vorhanden, so wird für das jeweilige Element geprüft, ob ein Algorithmus vom Nutzer definiert wurde, ansonsten wird die Standard-Weiterleitregel (FIFO) angewandt. Abschließend erfolgt eine Weiterleitung. Erreichen Token Aktionsknoten, so wird der Kontroller benachrichtigt. Ist der zugrunde liegende Prozess bereit, wird die Weiterleitung des `FlowItems` durch ein `forwardResponse` ausgelöst. Ist der Prozess hingegen blockiert, wird abgewartet, bis der Prozess den blockierten Zustand verlässt. Dann wird erneut eine Prioritätsregel ausgewertet, um eventuelle Prioritätsänderungen während der Blockade zu berücksichtigen.

Für die Erzeugung von **Zufallszahlen** bietet JAMES II eine Reihe von Zufallsgeneratoren. Beispielsweise wurden die Pseudo-Zufallsverfahren `Linear Congruential Generator`, `Mersenne Twister` und `Randu` in JAMES II implementiert. Paralleler Zugriff auf einen Generator ist konfliktfrei umgesetzt und die Zufallsperiode kann nach einer bestimmten Anzahl von generierten Zahlen neu initialisieren werden, um ein konstantes Niveau an Zufallszahlen über lange Laufzeiten hinweg zu erreichen. Auch verschiedene statistische Verteilungen (Normal, Uniform, Poisson, etc.) stehen zur Verfügung und können in die Zufallsgeneratoren eingebunden werden. JAMES II benutzt interne Heuristiken, um einen Generator mit höchster Güte je nach Anwendungsfall zu initialisieren.

In einem ersten Ansatz wird die in 7.1 beschriebene Systemarchitektur, der in 7.3 beschriebene Simulationsalgorithmus sowie die Integration des Modells in JAMES II umgesetzt. Modelle können somit

in TOPCASED erzeugt werden. Anschließend werden sie in das interne Modell übertragen und für JAMESII aufbereitet sowie simuliert. In der aus dieser Arbeit entstandenen Implementierung können aktuell einfache Warteschlangenmodelle, Prozesse und Ressourcen simuliert werden. Der Ansatz wird in einer zugrunde liegenden aktuellen Diplomarbeit umgesetzt und soll das im Kapitel 2.3 beschriebene strukturelle Metamodell und das im Kapitel 2.4 beschriebene Verhaltensmodell transformieren und simulieren können.

8 Anwendungen des Modellierungskonzeptes auf Domänen

In den folgenden Ausführungen werden Modellierungskonzepte spezieller Domänen des Untersuchungsgegenstandes betrachtet. Mit den Betrachtungen ist es zum einen möglich, zu überprüfen, ob sich das Modellierungskonzept der Dissertation für praktische Betrachtungen eignet, und zum anderen, das Modellierungskonzept domänenspezifisch um Besonderheiten zu erweitern. Neben den Betrachtungen der domänenspezifischen Modellierungseigenheiten erfolgt zudem eine Überprüfung der Notwendigkeit der Betrachtung spezieller Domänen wie der Unikatfertigung oder der Inlinefertigung von Solarzellen. Dadurch wird ein Überblick über die Einsatzmöglichkeiten des Modellierungskonzeptes gegeben.

Zuerst wird die Domäne der Halbleiterfertigung veranschaulicht, in der die diskrete eventgesteuerte Simulation weit verbreitet ist. Anschließend wird die prozessnahe Inlinefertigung von Solarzellen anhand eines praktischen Beispiels untersucht. Auch die Unikatfertigung wird betrachtet, die als Anwendungsbereich für diskrete Event Simulation in den letzten Jahren verstärkt diskutiert wurde. Abschließend werden die Domänen der Transportlogistik und Montage veranschaulicht.

8.1 Modellierung von Fertigungsprozessen in der Halbleiterindustrie

Die Herstellung von Halbleitern kann in die beiden Teilbereiche *Frontend* und *Backend* unterteilt werden. Beim *Frontend* findet die Herstellung und der Test der *Wafer* statt, beim *Backend* werden die *Wafer* zersägt, zu Bauelementen montiert und abschließend getestet. Die folgenden Betrachtungen erfolgen für die *Frontend*-Produktion. Seit vielen Jahren, besonders allerdings durch den Kostendruck der letzten Jahre wird versucht, diese Produktionsbereiche durch verstärkte Automatisierung und bessere Ablaufsteuerung effizienter zu gestalten. In der Backend-Entwicklung wurde dieser Prozess eher vollzogen. In modernen 300mm-Fertigungen werden die Ablaufsteuerungen vollautomatisch und regelbasiert eingesetzt. Jedoch lassen sich die dort verwendeten Verfahren nicht auf den Backend-Bereich übertragen (März et al., 2011, S. 50). Im Backend werden erhöhte Anforderungen an die Flexibilität gestellt, wodurch die Neuplanintervalle kürzer sind (vgl. Potoradi et al., 2002). Jedoch ist das Vorkommen von Zyklen geringer und für die Prozesse liegen konkrete Planzeiten vor (März et al., 2011, S. 50).

8.1.1 Modellierungskonzept in der Halbleiterfertigung

Das Programm Factory Explorer wurde speziell für die Modellierung und Simulation von Prozessen der Halbleiterfertigung entwickelt (Wright Williams & Kelly, 2014). In den folgenden Betrachtungen wurde das Modellierungskonzept des Factory Explorers und somit der Halbleiterproduktion analysiert. Es werden spezielle Begrifflichkeiten und deren mögliche Umsetzung erläutert.

Units, Lots und Batches

In der Halbleiterindustrie durchlaufen Produkte – gruppiert in *Lots* – den Produktionsprozess. Ein *Lot* besteht aus mehreren einzelnen *Units*. *Units* bzw. einzelne *Wafer,* sind die kleinste Einheit im Frontendbereich der Halbleiterproduktion. Mehrere *Lots* können zudem zu sogenannten *Batches* zusammengefasst werden. Maschinen können einzelne *Units*, *Lots* oder *Batches* bearbeiten. Maschinen haben eine Mindestanzahl und Höchstanzahl an Einheiten, die zugleich bearbeitet werden können. Wenn beispielsweise ein Ofen zum Erhitzen genutzt wird, kann es für die Wärmeverteilung nötig sein, eine bestimmte Mindestzahl an Einheiten zu bearbeiten.

Das Batch-Konzept

Die möglichen Maschinenbelegungen bei Batchmaschinen sind sehr komplex. Für die Modellierung spezieller Eigenschaften bei der Bearbeitung von *Batches* bietet der Factory Explorer daher sogenannte *Batch IDs*. So kann beispielsweise ein Ofen verschiedene Produktgruppen gleichzeitig bearbeiten. Jede Maschine, die *Batches* bearbeiten kann, besitzt eine Mindest- und Maximalgröße zum Bearbeiten von Produkten sowie eine *Batch ID*. Nutzt ein Prozess eine Maschine für seine Bearbeitung, muss auch für den Prozess eine *Batch ID* definiert sein. Anhand der *Batch ID* ist es dann möglich, festzulegen, welche Produkte oder Prozesse gemeinsam mit jeweils welcher Mengenanzahl in einer Maschine bearbeitet werden können. So kann bei der *Batch ID* der Maschine angegeben werden, ob nur *Lots* mit gleicher *Batch ID*, vom gleichen Prozess oder vom gleichen Produkt miteinander verarbeitbar sind. Zudem ist es möglich, in der *Batch ID* des Prozesses zu notieren, wie viele *Wafer* eine Maschine von dem jeweiligen Prozess maximal bearbeiten kann.

Rework

In der Halbleiterproduktion gibt es festgelegte Prüfvorgänge. Einer ist das sogenannte *Rework*. Nach bestimmten Prozessschritten kann es notwendig sein, zu testen, ob einzelne *Lots* oder *Units* wiederbearbeitet werden müssen. Ist das komplette *Lot* fehlerfrei, kann es zum nächsten Prozessschritt weitergeleitet werden, ansonsten wird getestet, ob das vollständige *Lot* oder nur bestimmte *Units* erneut zu bearbeiten sind. Ist ein *Rework* nötig, zerfällt das *Lot* in ein *Parent Lot* und ein *Child Lot*. In das *Parent Lot* werden alle Einheiten sortiert, die nicht erneut zu bearbeiten sind, in das *Child Lot* kommen die fehlerhaften. Das *Parent Lot* wird zum nächsten Prozessschritt weitergeleitet, während das *Child Lot* eine Reihe von Nachbearbeitungsschritten (*Rework Steps*) durchlaufen muss. Nach dem Durchlauf

der *Rework Steps* gelangt das *Child Lot* wieder zum Beginn der *Rework* Prozedur, bis kein *Rework* mehr notwendig ist.

Scrap

Ein anderer Prüfvorgang bei der Halbleiterproduktion ist das prüfen auf Ausschuss (*Scrap*). Erst wird geprüft, ob das ganze *Lot* zu verwerfen ist. Wenn das nicht der Fall ist, werden die einzelnen *Units* überprüft. Anschließend wird ein um die Anzahl der verworfenen *Units* reduziertes *Lot* an den nachfolgenden Prozessschritt weitergegeben.

Startzeiten

Bei der Modellierung von Produkten ist es möglich, ihnen eine Startzeit zuzuweisen, ab der sie verfügbar sind. Produkte ohne Startzeit sind bei Beginn der Simulation verfügbar.

Belade- und Entladezeiten sowie bedingte Arbeitszeiten von Operatoren

Die in diesem Abschnitt beschriebenen Eigenschaften sind im zugrundeliegenden Metamodell für Produktionsprozesse in der `ExtAssociation` schon berücksichtigt. Da sie aber selten von den betrachteten Programmen sowie Domänen genutzt wurden, wird ihre Verwendung hier noch einmal besonders hervorgehoben. Gerade Rüstzeiten (`transitionTimeDefinition`) sind bei den komplexen Maschinen in der Halbleiterfertigung ein großes Thema und werden daher zwingend als Belade- und Entladezeiten benötigt. Da bei bestimmten Prozessen Operatoren nur zum Beladen und Entladen benötigt werden, ist die `earlyReleaseDefinition` der `ExtAssociation` für die Domäne der Halbleiterfertigung relevant.

Erweiterte Fehler

Im definierten Metamodell für Produktionsmodelle können Fehler nach zeitlichem Auftreten modelliert werden. In der Halbleiterfertigung sind Fehlerraten zu finden, die nach der Anzahl der bearbeiteten Einheiten auftreten. Zudem kann ein Zeitpunkt für das frühst mögliche Auftreten von Fehlern definiert werden, ab diesem beginnt dann die übliche Berechnung über Fehlerintervallverteilungen.

8.1.2 Erweiterung des Metamodells

In den folgenden Ausführungen wird das Metamodell für Produktionsprozesse dieser Arbeit für die Halbleiterfertigung erweitert. Abbildung 8-1 zeigt das vollständig erweiterte Metamodell. Zudem werden die Erweiterungen der einzelnen Elemente besprochen und tabellarisch dargestellt. Das Stereotyp **`Entity`** wird um das Attribut `lotSize` erweitert. Das Attribut gibt an, aus wie vielen Einheiten (*Units*) ein *Entity* (*Lot*) besteht.

Tabelle 8-1: Definition erweitertes `Entity`

Attribut	Type	Syntax	Semantik
lotSize	Integer	0 ... 9999	Anzahl gibt an, aus wie vielen Units das als Lot bezeichnete Entity enthält.

Beim **`ArrivalProcess`** kann durch die Spezifikation des Attributes `arrivalStart` ein Zeitpunkt angegeben werden, der den `ArrivalProcess` aktiviert. Bestimmte `Entities` können also erst ab einem bestimmten Zeitpunkt in das System eingeschleust werden.

Tabelle 8-2: Definition erweiterter `Arrival Process`

Attribut	Type	Syntax	Semantik
arrivalStart	Tag-Monat-Jahr	Enumeration	Zeitpunkt, zu dem bestimmt wird, wann der Arrival Prozess startet.

Die Elemente **`SingleProcess`** und **`Mode`** werden durch etliche Eigenschaften erweitert. Das Attribut `processingUnit` gibt an, ob *Units*, *Lots* oder *Batches* verarbeitet werden. Die `processingTime` gibt nun die Prozesszeit für die entsprechende Größeneinheit an. Die `batchId` ist wie in 8.1.1 beschrieben notwendig, um mit Schlüsselwörtern anzugeben, welche Produkte oder Prozesse gemeinsam in einer Maschine bearbeitet werden können. Erhält die `batchID` den Wert `Process`, ist es möglich, gleiche Prozesse, bei `Product` Prozesse gleicher Produkte und bei `Operation` Prozesse mit gleicher `operationId` zu bearbeiten. Die Attribute `batchSizeMin` und `batchSizeMax` definieren die Mindest- und Maximalgröße eines *Batches*. Die `loadTimeDefinition` und `unloadTimeDefinition` geben eine für den Prozess konstante Belade- und Entladezeit an. Für die Angaben der Zeiteinheit und Verteilung erfolgt die Verwendung der schon bestehenden Attribute `processingTimeUnit` und `processingTimeDistribution`. Das Attribut `splitRecombine` gibt an, ob der Prozess *Lots* aufteilt (`split`), zusammenführt (`recombine`) oder einfach nur weiterleitet (`clear`). Wurde der Wert `split` angegeben, werden die eingehenden *Lots* nach ihrer Bearbeitung in *Lots* der Größe `batchSizeMin` aufgeteilt. Bei der Angabe des Wertes `recombine` werden aufgeteilte *Lots* nach der Bearbeitung durch den Prozess zur Größe `BatchSizeMax` zusammengeführt (vgl. Tabelle 8-3).

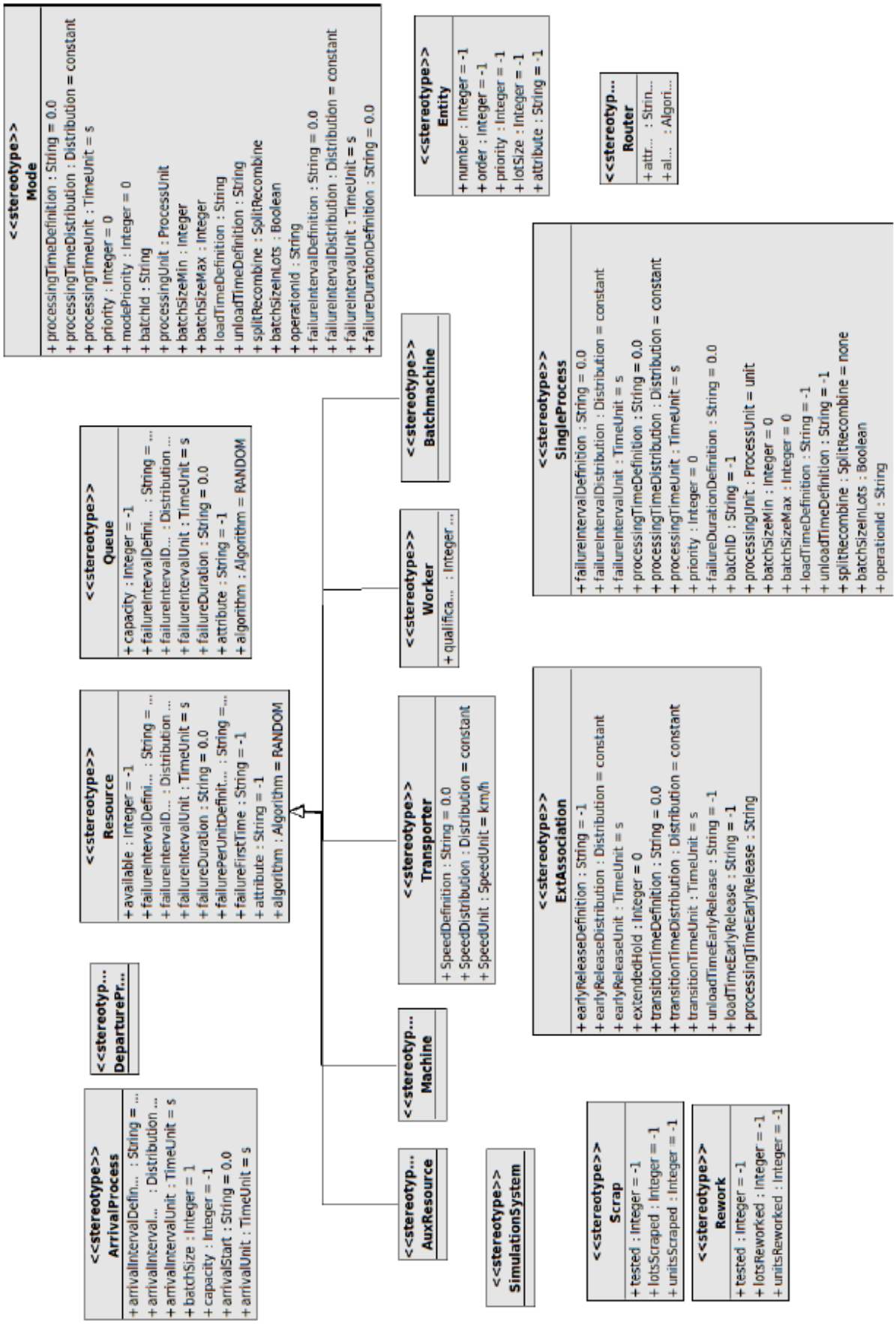


Abbildung 8-1: Für die Halbleiterfertigung erweitertes Metamodell

Tabelle 8-3: Definition der Erweiterungen für die Elemente `Process` und `Mode`

Attribut	Type	Syntax	Semantik
processingUnit	ProcessUnit	Enumeration	Angabe der Bearbeitungseinheit Unit , Lots oder Batches.
batchId	String	Process or Product or Operation	Gibt an, welche Prozesse zugleich beim Ausführen einer Maschine bearbeitet werden können.
operationId	String	ID	Beim Wert Operation bei batchID können Prozesse mit gleicher operationID zugleich in Maschine
batchSizeMin	Integer	0 ... 9999	Wie viele Units müssen mindestens in den Prozess, um ihn zu starten.
batchSizeMax	Integer	0 ... 9999	Wie viele Units können maximal in den Prozess
loadTimeDefiniti on	String	Vgl. Tabelle 3.9	Zeitwert zum Rüsten der Maschine.
unloadTimeDefin ition	String	Vgl. Tabelle 3.9	Zeitwert zum Abrüsten der Maschine.
splitRecombine	String	Clear or Split or Recombine	Definiert, ob Batch nach den Prozess gesplittet, zusammengesetzt oder einfach weitergeleitet wird.

Das Element **`Ressource`** wird um das Attribut `failurePerUnit` erweitert, das angibt, nach wie vielen bearbeiteten Einheiten ein Fehler auftritt. Als Verteilungsfunktion verwendet `failurePerUnit` die Funktion aus dem Wert `failureIntervallDistribution`. Zudem wurde der Wert `failureFirstTime` definiert, der angibt, wann ein Fehler erstmalig auftritt und ebenfalls die Verteilungsfunktion aus dem Wert `failureIntervallDistribution` sowie die Zeiteinheit aus dem Wert `failureIntervalTimeUnit` verwendet (vgl. Tabelle 8-4).

Tabelle 8-4: Definition der Erweiterungen für das Element `Ressource`

Attribut	Type	Syntax	Semantik
failurePerUnit	String	Vgl. Tabelle 3.9	Nach wie viel bearbeiteten Einheiten tritt ein Fehler auf.
failureFirstTime	String	Vgl. Tabelle 3.9	Nach welcher Dauer Tritt das erste Mal ein Fehler auf.

Mit den neuen Parametern `loadTimeEarlyRelease`, `unloadTimeEarlyRelease` und `processTimeEarlyRelease` kann für die **`ExtAssociation`** zwischen einem `SingleProcess` und einem assoziierten `Worker` angegeben werden, für wie viel Prozent der Arbeitszeit der `Worker` benötigt wird. Ist beispielsweise ein `SingleProcess` mit einer `Machine` und einem `Worker` verknüpft, gibt das Attribut an, wie lange der `Worker` an der `Machine` arbeitet (vgl. Tabelle 8-5).

Tabelle 8-5: Definition der Erweiterungen für das Element `ExtAsociation`

Attribut	Type	Syntax	Semantik
loadTimeEarlyRelease	String	0..100	Nach wie viel Prozent der Bearbeitungszeit kann assoziierter Worker freigegeben werden.
unloadTimeEarlyRelease	String	0..100	Nach wie viel Prozent der Bearbeitungszeit kann assoziierter Worker freigegeben werden.
processTimeEarlyRelease	String	0..100	Nach wie viel Prozent der Bearbeitungszeit kann assoziierter Worker freigegeben werden..

Das Element **`Scrap`** wurde neu hinzugefügt, um `Scraps` (Ausschusswahrscheinlichkeiten) modellieren zu können. Der Wert `tested` gibt an, mit welcher Wahrscheinlichkeit ein *Lot* auf Ausschuss geprüft wird. Der Wert `lotsScrapes` gibt die Wahrscheinlichkeit, mit der das gesamte *Lot* verworfen wird, an. Wird das *Lot* nicht verworfen, gibt `unitsScraped` an, wie groß der Anteil des Ausschusses bei den *Units* ist (vgl. Tabelle 8-6).

Tabelle 8-6: Definition der Erweiterungen für das Element `Scrap`

Attribut	Type	Syntax	Semantik
tested	Integer	0..100	Wie viel Prozent der Lots werden getestet.
lotsScrapes	Integer	0..100	Wie viel Prozent der getesteten Lots sind Ausschuss.
unitsScraped	Integer	0..100	Wie viel Prozent der getesteten Units sind Ausschuss.

Auch das Element **`Rework`** wurde neu in das Metamodell eingefügt. Mit dem Element ist es möglich, das einführend besprochene Rework-Muster nachzubilden. Im Attribut `tested` wird die Wahrscheinlichkeit definiert, mit der ein *`Lot`* auf `Rework` getestet wird. Für den Test des *Lots* wird in `lotsReworked` angegeben, mit welcher Wahrscheinlichkeit die Wiederbearbeitung des gesamten *Lots* erfolgt. Ist es nicht notwendig, das ganze *Lot* erneut zu bearbeiten, wird in `unitsReworked` der Anteil der erneut zu bearbeitenden *Units* angegeben (vgl. Tabelle 8-7).

Tabelle 8-7: Definition der Erweiterungen für das Element `Rework`

Attribut	Type	Syntax	Semantik
tested	Integer	0..100	Wie viel Prozent der Lots werden getestet.
lotsReworked	Integer	0..100	Wie viel Prozent der getesteten Lots sind wieder zu bearbeiten.
unitsReworked	Integer	0..100	Wie viel Prozent der getesteten Units sind wieder zu bearbeiten.

8.2 Modellierung der Inlinefertigung von Solarzellen

Die Produktion von Solarzellen, genauer gesagt Photovoltaikzellen, kann als spezielle Domäne der Halbleiterproduktion betrachtet werden. Am Weltmarkt werden jährlich durchschnittlich 30 bis 40 Prozent Wachstumsraten für Photovoltaik verzeichnet (vgl. Kopte, 2012). Trotzdem ist der Markt für Photovoltaik hart umkämpft. Aufgrund des intensiven Wettbewerbs besteht ein erheblicher Kostendruck auf die Hersteller von Photovoltaikanlagen (vgl. Miller/Kondruweit, 2012). Daher ist die kostengünstige Produktion der Solarzellen ein wichtiger Faktor, der in Zukunft wohl nicht an Bedeutung verlieren wird. In den folgenden Ausführungen wird die Modellierung eines konkreten Beispiels einer Linienfertigung von Solarzellen betrachtet (vgl. Pappert, 2010). Da die Produktion von Solarzellen als spezielle Domäne der Halbleiterproduktion verstanden werden kann, wird das im Kapitel 8.1 hergeleitete Metamodell für die Halbleiterfertigung verwendet. Es ist zu überprüfen, ob es möglich ist, die Produktion von Solarzellen mit diesem Metamodell zu modellieren. Das Metamodell kann anhand der speziellen Anforderungen erweitert werden.

Die Begrifflichkeiten ähneln denen aus Kapitel 8.1. *Lots* tragen beispielsweise in der Inlinefertigung die Bezeichnungen *Boxen* oder *Carrier*. Um die Begriffe anzupassen, werden einzelne Durchlaufobjekte, die oft auch *Wafer* genannt werden, in den folgenden Betrachtungen als *Units* bezeichnet und deren Ansammlung als *Lots*.

Die Abbildung 8-2 zeigt die vollständige Produktionsstrecke des Beispiels der Solarzellenfertigung. Zu Beginn der Strecke starten fertige Siliziumscheiben im Prozess AKL in das Produktionssystem ein. Nach Durchlaufen der Produktionsstrecke verlassen sie das System als Solarzellen, die in einer anderen Produktionsstrecke zu Solarmodulen montiert werden. Die betrachtete Fertigung unterteilt sich in vier identische Fertigungsstrecken, die sich jeweils in drei Segmente unterteilen lassen. Die vier Produktionsstrecken und deren Segmente sind durch sogenannte *Overheadbuffer* verbunden. Die *Overheadbuffer* entlasten einzelne Segmente einer Fertigungsstrecke, indem *Lots* gespeichert oder auf anderen Fertigungsstrecken umgeleitet werden können. So ist es möglich, Ausfälle durch das Auftreten von Fehlern oder Wartungsarbeiten auszugleichen. In den folgenden Ausführungen wird die Ferti-

gungsstrecke betrachtet und versucht, die Eigenheiten der einzelnen Elemente herauszuarbeiten und im Kontext des Modellierungskonzeptes dieser Arbeit darzustellen.

Es ist nicht möglich, die Modellierung der *Overheadbuffer* zu untersuchen, da deren Funktionsweise in keiner einsehbaren Arbeit veröffentlicht wurde. Insgesamt besteht jede der vier Fertigungsstrecken aus zwanzig Elementen, die in den folgenden Ausführungen betrachtet werden. Die grünen Elemente der Abbildung 8-2 symbolisieren Prozesse, die *Units* zu *Lots* zusammenführen oder die *Units* eines *Lots* entpacken oder in mehrere *Lots* aufteilen. Die grauen Elemente symbolisieren Prozesse, die *Units* oder *Lots* bearbeiten. Die gelben Elemente symbolisieren die Prozesse des *Overheadbuffers,* die wie ausgeführt nicht detailliert beschrieben werden können. Stattdessen erfolgt deren Darstellung durch einfache Warteschlangen.

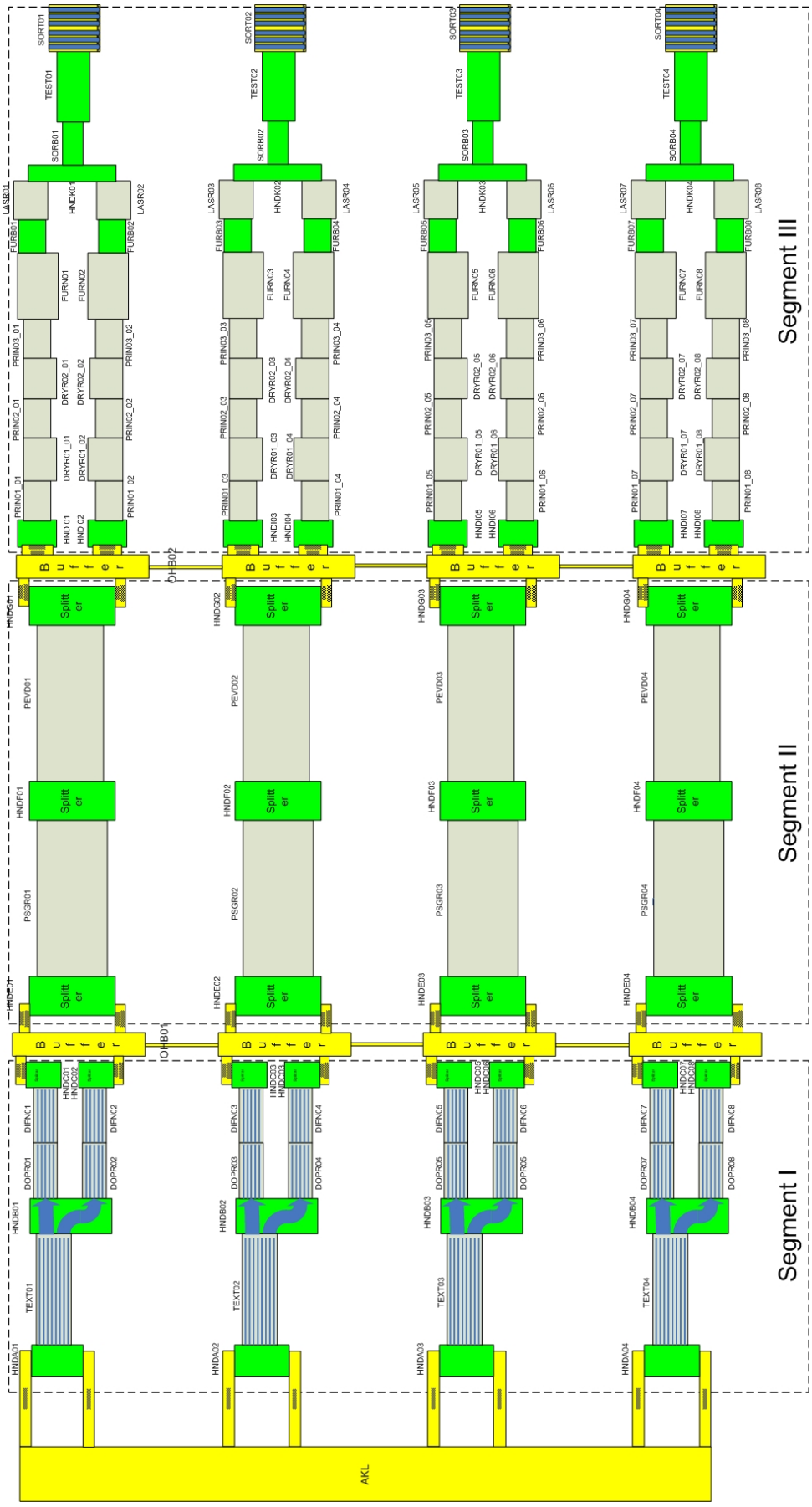


Abbildung 8-2: Produktionsstrecke einer Linienfertigung von Solarzellen (vgl. Pappert, 2010)

8.2.1 Modellierung der Komponenten

Die folgenden Betrachtungen beschäftigen sich mit der detaillierten Modellierung der drei Segmente. Zuerst wird jeweils das vollständige Segment und der Ablauf der ihm zugeordneten Prozesse vorgestellt. Anschließend werden die Elemente des jeweiligen Segments in ihrer Modellierung veranschaulicht. Es ist auch möglich, die verschiedenen Elemente der Segmente als Maschinen oder *Tools* zu bezeichnen. *Units* können auch als *Wafer* und *Lots* als *Boxen* oder *Carrier* bezeichnet werden. Während *Boxen Units* gestapelt sammeln, sind *Carrier* eine ebene Sammelplattformen für *Wafer*. Diese Dissertation betrachtet beide Formen als *Lots*.

Segment I

Im ersten Segment starten die fertigen Siliziumscheiben durch das Element *Materialhandler A (HNDA)* in die Fertigungsstrecke ein und wandern dann zum *Tool Texturing (Text)*. Die *Wafer* kommen in *Lots* beim *Materialhandler* an und werden von ihm als *Units* weitergegeben. Die *Wafer* wurden vor Betreten der Produktionsstrecke aus einem Silizium-Ingot gesägt und haben daher eine raue Fläche. In Text fahren die *Wafer* durch ein Säurebad, um sie dadurch zu glätten. Anschließend werden die *Units* vom *Tool Materialhandler B (HNDB)* in *Boxen* gelegt, gelagert, auf verschiedene *Lots* geteilt und an *Doping (DOPG)* weitergegeben. Die *Units* durchlaufen nun die beiden Prozessschritte *Doping* und *Diffusion,* in denen sie mit Ionen dotiert werden. Abschließend an Segment I werden die einzelnen *Wafer* in eine *Box* gelegt und im *Overheadbuffer* gelagert (vgl. Abbildung 8-3).

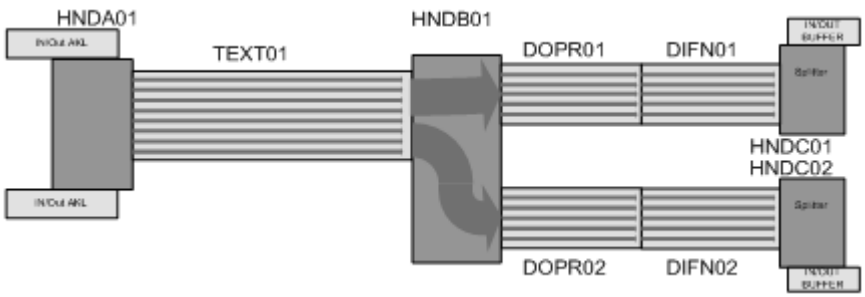


Abbildung 8-3: Zusammenhängende Darstellung des Segment I (vgl. Pappert, 201, S. 10)

Die folgenden Betrachtungen widmen sich den einzelnen *Tools* des ersten (vgl. Abbildung 8-4). Das *Tool Materialhandler A* bekommt *Wafer* in *Lots* durch die Förderbänder D und C. Die Elemente A und B sind Greifarme, die die ankommenden *Wafer* den *Lots* entnehmen und auf die Förderbänder E und F legen. Diese transportieren die *Wafer* zum nächsten *Tool*. Bei den *Tools Materialhandler C* und *Materialhandler B* ist ein ähnlicher Ablauf zu finden. In der Maschine *Materialhandler C* werden die *Tools* vom Inputförderband B durch den Greifarm A in den *Lots* C zum *Overheadbuffer* gebracht. Das *Tool Materialhandler B* bekommt *Wafer* von den Förderbändern E und D und befördert sie durch die Greifarme B und C auf das Förderband G. Ein Teil der *Wafer* wird durch das Förderband F zum Greifarm C transportiert, der die *Wafer* auf das Förderband H oder in den Puffer J transportiert.

Jedes Förderband kann eine Länge und Geschwindigkeit und einen Abstand für die *Wafer* als Attribute besitzen. Die Greifarme haben eine `Pickertime` in Sekunden, die ausdrückt, wie viele Sekunden sie für einen *Wafer* zum Transport benötigen, als Attribut. Zudem wird die Anzahl der *Wafer* in einem *Lot* als Boxengröße, die Pufferzahl in *Wafer* und die Boxenwechselzeit in Sekunden angegeben. Aus diesen Daten können die Werte Durchsatz in *Wafer* pro Sekunde, Taktzeit in Sekunden pro *Wafer* und maximaler Durchsatz in *Wafer* pro Stunde berechnet werden. Zudem hat ein *Tool* die Fehler Attribute Bruch in Prozent, Bruchbehältergröße in *Wafer* und Bruchbehälterwechsel in Sekunden.

Die Prozesse Texturing (TEXT), Diffusion (DIFN) und DOPG haben eine Bandlänge, Bandgeschwindigkeit, einen Durchsatz in *Wafer* pro Sekunde und einen Maximaldurchsatz in *Wafer* pro Stunde. Zudem haben die *Tools* die genannten Fehlerattribute.

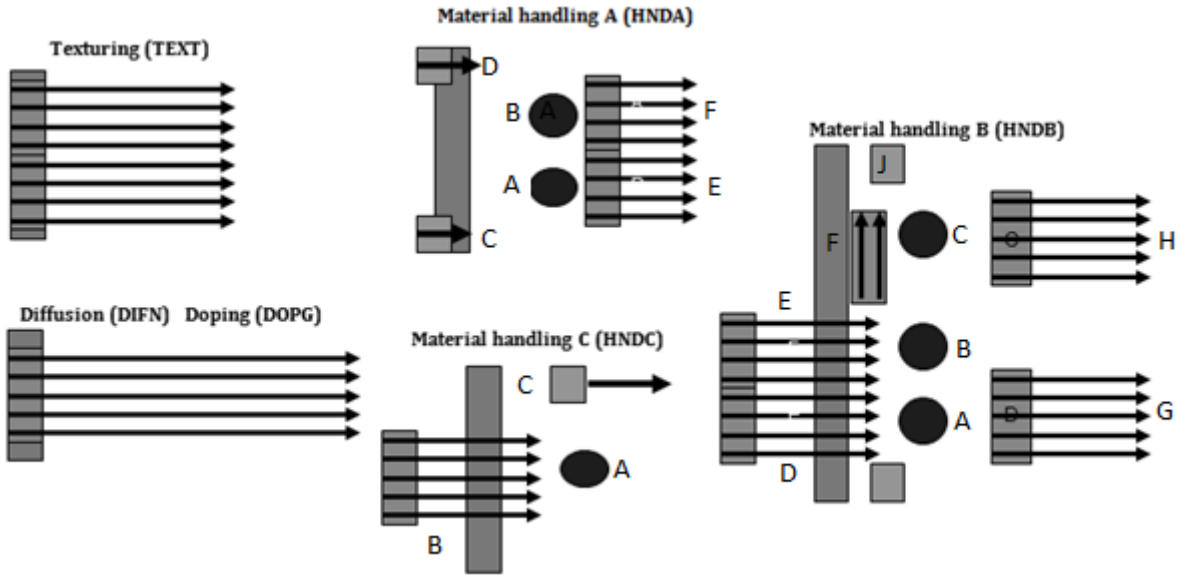


Abbildung 8-4: Darstellung der einzelnen Elemente des Segment I (Pappert, 2010, S.16ff.)

Segment II

Im zweiten Segment werden *Lots* vom *Materialhandling Tool E (HNDE)* vom *Overheadbuffer* empfangen und als einzelne *Units* dem *Tool PSG removal (PSGR)* übergeben. Im *PSG removal* durchqueren die *Wafer* einen weiteren Ätzschritt. Anschließend empfängt der *Materialhandler F (HNDF)* die *Units*, lagert sie in Boxen und gibt sie an das *Plasma Etch Chemical Vapor Deposition Tool (PEVD)* weiter. Die Übergabe vom *Materialhandler F* erfolgt mit der Hilfe eines sogenannten *Carrier*, in dem die *Wafern* als *Lot* geladen und an das Folgetool weitergegeben werden. Das *PEVD Tool,* das aus mehreren Kammern besteht, beschichtet die *Wafer* dann. Abschließend werden die *Wafer* im *Materialhandler F* in eine *Box* gelegt und im *Overheadbuffer* gelagert (vgl. Abbildung 8-5).

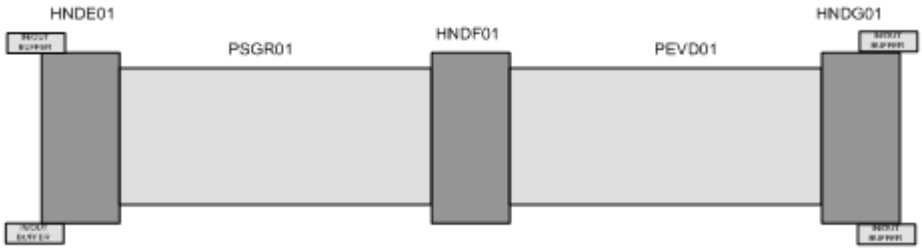


Abbildung 8-5: Zusammenhängende Darstellung des Segment II (Pappert, 2010, S. 11)

Es ist möglich, den *Materialhandler E (HNDE)* mit denselben Attributen wie die *Materialhandler* aus Segment I abzubilden. Das *Tool PSG removal* (PSGR) kann genau wie die Prozesse TEXT, DIFN und DOPG des ersten Segments modelliert werden.

Das *Tool Materialhandler F* (HNDF) startet durch die Fließbänder D und C *Lots* ein, die von den Greifarmen B und A an den *Carrier G* gegeben werden. Ist der *Carrier* voll, kann aber nicht weiterbewegt werden, werden die *Lots* in den Puffern F und E eingelagert. Die Puffer F und E haben eine Warnschwelle. Wenn diese erreicht ist, werden Signale gegeben, um keine *Lots* mehr in das Segment der Fertigungsstrecke einzulagern. Zur Modellierung des *Tools* wird das Attribut Warnschwelle (`threshold`) für das Element Warteschlange und alle Elemente, die Ein- und Ausgangspuffer haben, ergänzt. Das *Tool Material Handler G* (HNGD) kann ähnlich modelliert werden und befördert über die Greifarme A und B *Units* von den eingehenden Prozesskammern in *Lots* des *Overheadbuffers*.

Das PEVD besteht aus fünf Prozesskammern. Eine Prozesskammer wird durch eine Anzahl von *Lots* , einem sogenannten *Batch,* gefüllt. Während in der mittleren prozessiert wird, sind die vorderen und hinteren beiden Prozesskammern zum Lagern da. In Kammer C prozessierte *Lots* müssen nach dem Prozess sofort die Kammer verlassen, da sie sonst verbrennen. Um dies sicherstellen zu können, ist es nur erlaubt, *Lots* in C einzulagern, wenn die Prozesskammer D frei ist. Es muss also ein Attribut zugelassen werden, dass es erlaubt, einen Prozess erst zu starten, wenn ein anderer frei ist (`freeRelation`). In den Kammern A und B werden die *Lots* vorbearbeitet und können dann weitergeleitet werden oder weiter im Prozess lagern. Somit haben A und B keine modellierungstechnischen Besonderheiten (vgl. Abbildung 8-6).

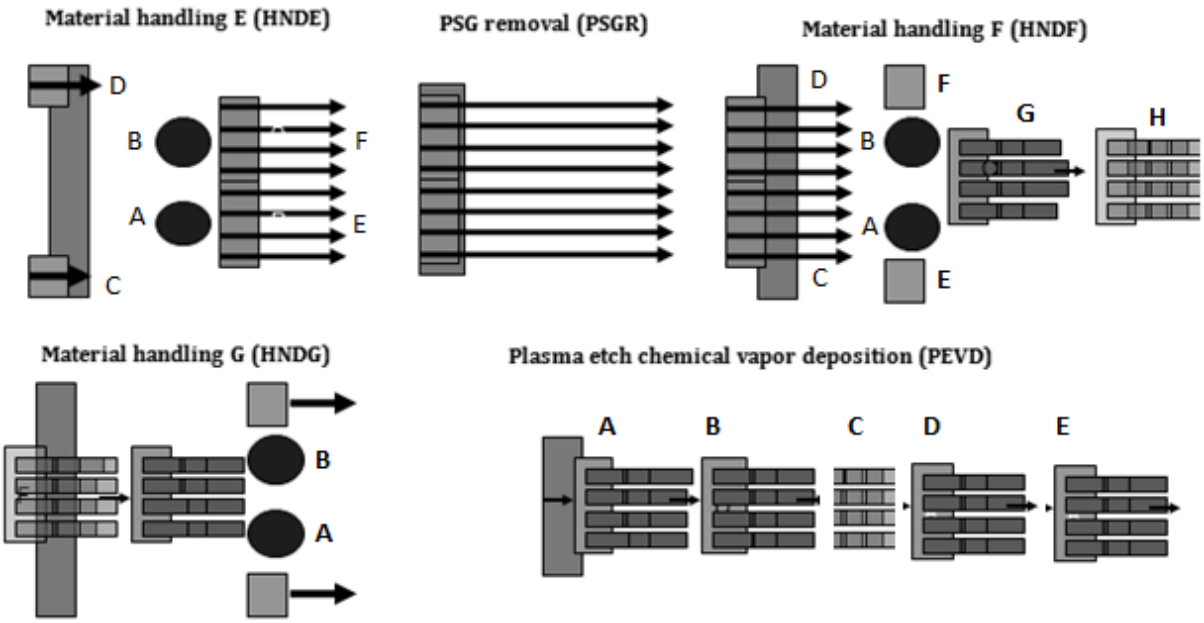


Abbildung 8-6: Darstellung der einzelnen Elemente des Segment II (Pappert, 201, S. 20ff.)

Segment III

Im dritten Segment werden die *Lots* vom *Materialhandler* I (HNDI) aus den *Boxen* des *Overheadbuffers* auf zwei *Carrier* für die taktgesteuerte Weitergabe auf zwei Folgebahnen verteilt. Nun erfolgt die Weitergabe der *Wafer* an eine Reihe von Folgeprozessen, die sie im Takt ablaufen. Zuerst bearbeiten *Printer* die *Wafer*, die dann zum *Dryer* wandern. Die *Printer* tragen unterschiedliche Metallisierungsebenen auf. Im *Tool Dryer* werden die *Wafer* getrocknet. Diese Abfolge wiederholt sich, bis die *Wafer* dann abschließend in einen *Furnace* erhitzen. Im *Furnace* wird der Takt beendet und die *Units* wandern auf einen Fließband weiter. Durch den Übergang von der Takt- zur Fließbewegung wird nach den *Furnace* ein Puffer benötigt (*Furnacebuffer*). Fließbänder leiten nun einzelne *Units* durch die Laserisolation. Dort findet eine Kantenisolation statt, in welcher der Laser einmal um die Kanten der *Wafer* fährt. Der nächste Schritt fügt die *Wafer* der beiden Prozessstrecken durch den *Materialhandler K* (HNDK) zusammen und die Fortbewegung wird wieder auf Taktzeit geregelt. Anschließend kommt die *Wafer* in *Sortbuffer* (SORB) und werden dann im *Celltest* (TEST) auf ihre Qualität getestet und in *Sorting Tool* (SORT) anhand dieser Qualität sortiert (vgl. Abbildung 8-7).

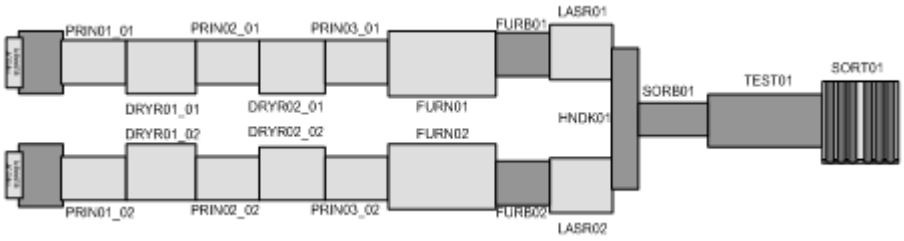


Abbildung 8-7: Zusammenhängende Darstellung des Segment III (Pappert, 2010, S. 11)

Die *Tools* des dritten Segments enthalten zum Großteil keine modellierungstechnischen Neuerungen. Die *Materialhandler I* und *K* können – wie die *Materialhandler* der anderen Segmente – modelliert werden. Der *Furnace* und die Laser-Isolation können wie das *Tool* TEXT des ersten Segments modelliert werden. Der *Furnacebuffer* und *Sortbuffer* stellen einfache *Queues* dar. Die *Tools Printer* und *Dryer* sind taktgesteuert. Modellierungstechnisch gibt es für die Taktsteuerung keine Besonderheiten– abgesehen davon, dass die *Tools* mit einer gleichen konstanten Bearbeitungszeit prozessieren. Das *Sorting Tool* ordnet die *Wafer* nach ihrer Produktqualität, dies ist mit einer benötigten Prozesszeit und anschließenden Oder-Verknüpfungen zu modellieren (vgl. Abbildung 8-8).

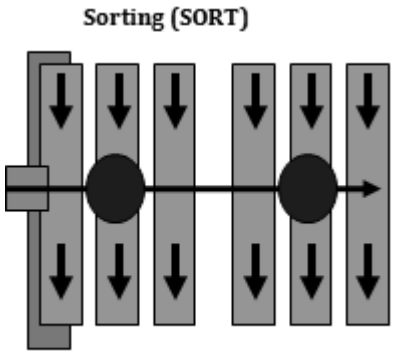


Abbildung 8-8: Darstellung des Elementes *Sorting* des Segment III (Pappert, 201, S. 23ff.)

8.2.2 Erweiterung des Metamodells

Die *Wafer* und *Lots* können durch das `Entity` und das Attribut `lotsize` dargestellt werden. Um die verschiedenen Bearbeitungsprozesse der *Tools* darstellen zu können, müssen neue Elemente und Attribute in das Metamodell eingefügt werden.

Ein Greifarm befördert einzelne *Units* aus einem *Lot* von einen Vorgängerelement zu einem Nachfolgeelement. Der Greifarm dient also dazu, die Anzahl der enthaltenen *Units* des Durchflussobjektes zu ändern. So können *Units* in *Lots*, *Lots* in *Units* und *Lots* in *Lots* verschiedener Größe transformiert werden. Da das Element Greifarm zu viele Attribute besitzt, wird es als neues Element neben dem Prozess definiert und als **`Picker`** bezeichnet. Als Attribute benötigt er die Zeit (`pickerTime`), die benötigt wird, um eine *Unit* zu befördern, die Anzahl (`inputLotSize`), die definiert, wie viele

Units ein eingehendes *Lot* enthält, sowie die Anzahl der *Units*, die ein ausgehendes *Lot* enthält (`outputLotSize`). Wenn eingehende *Units* in ein *Lot* gelegt werden sollen, besteht zudem eine Zeit (`outLotSwitchTime`) für das Wechseln des Lotbehälters, wenn dieser gefüllt ist. Wenn die *Units* dem `Picker` in *Lots* betreten, kann auch eine Zeit zum Wechsel eines leer gegriffenen *Lots* bestehen (`InLotSwitchTime`). Zudem kann ein `Picker` auch einen Eingangspuffer und einen Ausgangspuffer mit den Kapazitäten `inputBufferSize bzw. outputBufferSize` besitzen (vgl. Tabelle 8-8).

Tabelle 8-8: Definition des Elements `Picker`

Attribut	Type	Syntax	Semantik
pickerTime	Integer	0 ... 9999	Zeit,die der Picker benötigt um eine Unit vom Eingang zum Ausgang zu befördern.
inputLotSize	Integer	0 ... 9999	Anzahl der Units in einen Eingangs-Lot. Ist die Size auf 1 kommen Units in den Eingang.
outputLotSize	Integer	0 ... 9999	Anzahl der Units in einen Ausgangs-Lot. Ist die Size auf 1 kommen Units in den Ausgang.
inLotsSwitchTime	Integer	0 ... 9999	Zeit für den Wechsel eines leeren eingangs Lots in Sekunden.
outLotSwitchTime	Integer	0 ... 9999	Zeit für den Wechsel eines leeren ausgangs Lots in Sekunden.
inputBufferSize	Integer	0 ... 9999	Größe des Eingangspuffer.
outputBufferSize	Integer	0 ... 9999	Größe des Ausgangspuffers.

Die **Fließbänder**, die für die verschiedenen Bearbeitungsschritte eingesetzt werden, haben die Attribute Fließbandlänge, Geschwindigkeit und Breite. Damit ist es möglich, darzustellen, wie viele *Units* auf einem Fließband in welcher Zeit befördert werden können. Für diesen Sachverhalt ist das Element `Prozess` so zu erweitern, dass es auch Fließbandprozesse darstellen kann. Die Attribute `conveyerSize` und `processingTime` stellen dar, wie viele *Units* auf das Fließband können und wie lange eine *Unit* benötigt, um das Fließband zu überqueren. Können mehrere *Lots* oder *Units* in einer Reihe zur gleichen Zeit das Fließband überqueren, wird dies im Attribut `batchSize` angegeben. Ist der Wert der `batchSize` größer als eins, wird angegeben, wie viele *Batches* das Fließband fasst. Zudem ist es möglich, bei Fließbändern deren Maximaldurchsatz (`maxUnitsPerHour`) in *Wafer* pro Stunde anzugeben (vgl. Tabelle 8-9).

Tabelle 8-9: Erweiterung des Elements `Process`

Attribut	Type	Syntax	Semantik
conveyerSize	Integer	0 ... 9999	Wenn der Wert größer 1 ist, wird ein Fließband dargestellt und size bestimmt die Kapazität.
maxUnitsPerHour	Integer	0 ... 9999	Wie viele Units können maximal auf einem Fließband bearbeitet werden.

Die Warteschlange und alle Elemente die einen Eingangs- oder/und Ausgangspuffer besitzen (*Picker, ArrivalProcess*) werden um das Attribut **Warnschwelle** (`threshold`) ergänzt. Wenn die Warnschwelle erreicht ist, werden keine *Entities* mehr in das System eingeschleust (vgl. Tabelle 8-10).

Tabelle 8-10: Erweiterung der Elemente `Picker, ArrivalProcess` und `Queue`

Attribut	Type	Syntax	Semantik
threshold	Integer	0 ... 9999	Wenn die Warnschwelle des Puffers erreicht ist, werden keine Elemente mehr in das System geschleust

Im *Tool* PEVD des zweiten Segments werden Prozesskammern eingeführt. Bei der Frontend-Bearbeitung von *Wafern* gibt es zahlreiche Batchmaschinen, die zum Teil aus verschiedenen Prozesskammern bestehen. Die Prozesskammern können mit einer Fülle an Eigenschaften belegt sein. In dieser Arbeit werden die verschiedenen *Batchtools* und ihre Prozesskammern in der *Wafer-Frontend*-Produktion nicht tiefergehend untersucht. Somit ist es möglich, Prozesskammern einfach mit dem Element Maschine zu modellieren. Um das betrachtete Szenario modellieren zu können, ist es aber notwendig, eine Prozesskammer erst dann aktivieren zu können, wenn eine andere frei ist. Um dies modellieren zu können, wird das Element `ExtAssociation` um das Attribut `freeRelation` erweitert. Das Attribut kann zwischen zwei Prozessen gesetzt werden und erlaubt es den vorhergehenden, nur dann zu starten, wenn der folgende gerade arbeitet oder frei ist (vgl. Tabelle 8-11).

Tabelle 8-11: Erweiterung der `ExtAssociation`

Attribut	Type	Syntax	Semantik
freeReleation	Boolean	1/0	Erweiterte Assoziation zwischen zwei Maschinen. Ist der Wert auf 1 gesetzt, kann der erste der beiden Prozesse erst starten wenn der spätere Prozess arbeitet. Ist das Attribut auf 0 gesetzt, kann der erste der beiden Prozesse erst dann arbeiten, wenn der spätere Prozess leer ist.

Weiterhin konnte dem Szenario eine besondere **Fehlerspezifikation** entnommen werden. Jedes Element kann einen prozentualen Bruch haben (`failurePerUnit`), der im Metamodell für Halbleiterprozesse im Element *Ressource* angegeben wird und im `SingleProcess`, `Picker` und der `Queue` erweitert werden muss. Zudem können bei einem Prozess Bruchbehälter notwendig sein, die, wenn sie voll sind, gewechselt werden müssen. Daher erfolgt eine Erweiterung der Elemente `SingleProcess`, `Conveyor` und `Picker` um die Attribute `scrapContainerSize` und `scrapContainerChange` (vgl. Tabelle 8-12).

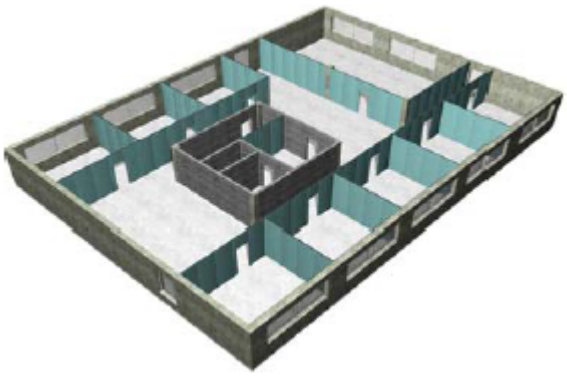
Tabelle 8-12: Erweiterung der Elemente `Picker, ArrivalProcess` und `Queue`

Attribut	Type	Syntax	Semantik
failurePerUnit	Integer	0..100	Wie viel Prozent der Lots werden beschädigt.
scrapContainerSize	Integer	0..100	Größe des Behälters der Ausschuss aussondert..
scrapContainerChange	Integer	0..9999	Benötigte Zeit zum Ausleeren des Behälters in Sekunden

8.3 Modellierung von Unikatprozessen, insbesondere im Bauwesen

Als Untersuchungsgegenstand war die Modellierung von Unikatprozessen in Deutschland lange Zeit weitgehend unbekannt. Erst 2007 fand eine erste Tagung zum Thema Simulation in der Bauwirtschaft in Kassel statt. Seit dem hat sich eine Arbeitsgruppe in der Arbeitsgemeinschaft Simulation (ASIM) Simulation in Produktion und Logistik gebildet, die sich mit dem Thema der Simulation von Unikatprozessen beschäftigt. Anfang 2011 fand der zweite *Workshop* „Simulation von Unikatprozessen – Neue Anwendungen aus Forschung und Praxis“ statt, dessen Besucherzahl einen deutlichen Anstieg des Interesses und Arbeitsfortschritt aufzeigte (Franz, 2011, S. 8). Die Domäne befindet sich auch im Interessengebiet dieser Arbeit und wird in den folgenden Ausführungen betrachtet. Die untersuchten Probleme der Unikatprozesse können im Kontext der Modellierung und im Kontext der Simulationsmodelle wie Produktionsprozesse eingeordnet werden und erweiterten lediglich den Anwendungsbereich des Untersuchungsgegenstandes (vgl. Kapitel 1.2).

Pietsch macht darauf aufmerksam, dass es eine Vielzahl von Simulationswerkzeugen gerade im Bereich der Materialfluss- und Logistiksimulation gibt. Aber nur wenige einen universellen Anspruch erheben können und es so gut wie keine für die Unikatfertigung gibt (Pietsch, 2011, S. 2 f.). Er vertritt die Meinung, dass ein solches Werkzeug über branchenspezifische Bibliotheken verfügen muss und vom Anwender parametrisierbaren Simulationsbausteinen, die im Optimalfall vom Anwender flexibel modifiziert und erweitert werden können (Pietsch, 2011, S. 16). Als Vorteile wird von der Arbeitsgruppe für Unikatprozesse die Sicherheit bei Planungsentscheidungen durch Test von Alternativen und eine daraus resultierende effizientere Planung gesehen. Zudem wird angenommen, dass der Aufwand für die Realisierung reduziert werden kann, wodurch die Kosten sinken und die Qualität der Ausführung steigt (Franz, 2010. S. 13). Beißert hat den Innenausbau eines Beispielgeschoßes mit neun Räumen durch Simulation des Ablaufes der Prozesse optimiert. Dabei simulierte sie mit den vier verschiedenen Ordnungsstrategien längste Schritte zuerst, kürzeste Schritte zuerst, viele Nachfolger zuerst und wenige Nachfolger zuerst. Innerhalb der Ergebnisse ergaben sich Differenzen von bis zu dreizehn Prozent (Beißert, 2010, S. 57).



Beispielgeschoss (30,52m x 20,93m)

- 13 Trockenbauwände
- 9 Räume
- 13 Fensterbänke
- 9 Innentüren

Abbildung 8-9: Grundriss einer Testetage eines 4-stöckigen Hauses (Beißert, 2010, S. 54)

Die Modellierung und Simulation von Unikatprozessen ist ein aktuelles Thema, welches in Zukunft an Bedeutung gewinnen könnte. Die folgenden Ausführungen sollen die Eigenheiten von Unikatprozessen erarbeiten, zudem wird überprüft, ob es möglich ist, Unikatprozesse mit dem Konzept dieser Arbeit abzubilden.

8.3.1 Definition der Unikatprozesse und Spezifizierung ihrer Eigenheiten im Kontext der Modellierung

Unter Unikatprozessen werden das Bauwesen, der Schiffsbau und der Anlagenbau bei der Errichtung von Industrieanlagen verstanden (Franz, 2011, S. 9). Pietsch unterscheidet zwischen einzigartigen Teilprozessen und Gesamtprozessen, die er jedoch beide als Unikatprozesse klassifiziert. Ein Unikat ist seiner Definition nach „das Ergebnis einer individuellen oder einzigartigen Anordnung und Parametrisierung von Prozessen“ (Pietsch, 2011, S. 15 f.). Fleming und Wach (2010, S. 121) definieren Unikatprozesse als „individuelle Anordnung von Ressourcen zur Erfüllung eines spezifischen Zwecks unter Beachtung spezieller Standort- und Rahmenbedingungen, dessen exakte Nachbildung nicht realisierbar ist“. Die Arbeitsgruppe „Unikatprozesse“ ermittelte folgende Besonderheiten für Unikatprozesse „

- ortsveränderliche Arbeitssysteme mit technologischen Einschränkungen,
- Einmaligkeit der Planung und Ausführung,
- viele Störungen und Parameter mit stochastischen Eigenschaften,
- viele unterschiedlich agierende Akteure und Beteiligte,
- vielfältige zwingende und sinnvolle Abhängigkeiten in den Prozessen,
- Erstreckung über lange Zeiträume.“ (vgl. Franz, 2010, S. 7)

Die einzelnen Ausführungen bei Unikatprozessen sind sehr individuell und können oft nur einmal ausgeführt werden. Die Randbedingungen können sich während des Prozesses ändern. Jedoch sind die

Grundstrukturen oft ähnlich. Durch räumliche und klimatische Umgebungseinflüsse sind Unikatprozesse besonders vom Zufall abhängig und daher stochastisch zu modellieren. Zudem sind im Hochbau und im Schiffsbau die Zahl der verschiedenen Akteure und die Abhängigkeiten besonders ausgeprägt und vielfältig (Franz, 2011, S. 10).

Die Arbeitsgruppe hat Bauprozesse in Planungsprozesse, Fertigungsprozesse, Nutzungsprozesse und Rückbauprozesse unterteilt. In dieser Arbeit sollen die Planungs- und Fertigungsprozesse betrachtet werden, deren Charakter dem mittelfristigen und strategischen Charakter vom Untersuchungsgegenstand im Aufgabenfeld der Produktion ähneln (vgl. Kapitel 1.2). Des Weiteren wurde von der Arbeitsgruppe Unikatprozesse ein Vergleich von Bauprozessen und stationärer Produktion aufgestellt (vgl. Tabelle 8-13).

Tabelle 8-13: Vergleich der Bauprozesse mit stationären Produktionsprozessen (vgl. Franz, 2010. S. 7)

Kriterium	Stationäre Produktion	Baustelle
Fertigungsplanung	Durch Serienfertigung festgelegt, einmalige Planung für viele Ausführungen	Durch unterschiedliche Fertigungsbedingungen, einmalige Planung für einmalige Ausführung.
Produktionsort	Ortsgebunden an einer Produktionsanlage, geschützt gegen Witterungen	Wechselt ständig sowohl mit dem Projekt als auch während der Produktion eines Projektes, stark witterungsabhängig.
Herstellung	Festgelegter getakteter Ablauf, fast ohne Behinderungen	Meist gestörter Ablauf wegen vieler, unvorhersehbarer Behinderungen bedingt durch Einzigartigkeit des Projektes.
Produktionsdauer	Relativ kurz sowohl für eine Serie als auch für einen Auftrag, vorab gut planbar	Nicht planbar, ständige Störungen im Terminplan, erstreckt sich über Jahre (unsichere Kostenbedingungen).

Die Einmaligkeit und Einzigartigkeit der Unikatprozesse erscheint zunächst schwer erfassbar zu sein. Jedoch wird bei der Definition auch von einer einzigartigen Aneinanderreihung und Parametrisierung gesprochen. Dieser Sachverhalt ist mit einem Simulationswerkzeug mit Sicherheit zu bewältigen. Wichtig zur Beantwortung der Frage, ob Unikatprozesse mit dem vorgestellten oder einem anderen Modellierungskonzept erfasst werden können, ist es, wie zahlreich und individuell die einzelnen Prozessschritte sind. Den gesammelten Erfahrungen dieser Arbeit nach sind die spezifischen Attribute von Prozessschritten meist semantisch sehr ähnlich oder als gleich zu klassifizieren. Eine Aussage der Arbeitsgruppe lässt diesen Befund zu: „Ein Unikat ist zwar einmalig, aber kann aus vielen durchaus sich wiederholenden Teilaufgaben und -prozessen bestehen“ (Franz, 2010. S. 15).

Dass im Bauwesen viele Störungen und stochastische Eigenschaften vorkommen, ist kein Problem für die Modellierung der Prozesse, doch stellt dieser Sachverhalt spezielle simulationsspezifische Anforderungen. Auch die Vielzahl an Akteuren und Beteiligten stellt für die Modellierung keine besondere Schwierigkeit dar und spricht eher für die Nutzung von Beschreibungssprachen, um die Komplexität der Sachverhalte bewältigen zu können. Bei der Vielfältigkeit der Abhängigkeiten ist wieder die Anzahl der semantisch differenzierten Sachverhalte interessant. Wie bei der Spezialisierung der Prozess-

schritte wird durch die praktischen Erfahrungen die Vielfältigkeit der semantischen Klassifikation der Abhängigkeiten als überschaubar angenommen. Ein vielfältiges Auftreten von semantisch ähnlichen Abhängigkeiten stellt kein Problem für die Modellierbarkeit der Unikatprozesse dar. Die Erstreckung der Projekte über längere Zeiträume ist auch kein modellierungsspezifisches Problem.

Beißert trägt in ihrer Arbeit *Constraints* für Bauprozesse zusammen. Dabei nennt sie (vgl. Beißert, 2010, S. 50):

- Reihenfolgerestriktionen zwischen Bauprozessschritten,
- Anzahl und Qualifikation von Arbeitsmitteln und Arbeiter,
- Material unter der Berücksichtigung von vorhandenen Lagerflächen,
- Sicherheitsvorschriften für Arbeiter und Arbeitsmittel,
- Etablierte Ausführungsreihenfolgen als *Soft-Constraint,*
- Ressourcenunterbrechungen.

Die ersten beiden Punkte sind Restriktionen, die auch in der Produktion vorkommen. Die Sicherheitsvorschriften legen sicher *Constraints* fest, wie beispielsweise, eine Mindestzahl von Arbeitern an bestimmten Prozessen oder welche Maschinen bei bestimmten Prozessen benötigt werden. Für die genannten Anwendungen würden die Lösungen des Modellierungskonzeptes der Arbeit ausreichen. Auch Ressourcenbedingungen sind im Modellierungskonzept enthalten. Optional ist es möglich neue Attribute für die `ExtendedAssociation` des Modellierungskonzeptes zu definieren. Für etablierte Ausführungsreihenfolgen könnten einfach mehrere Prozesse zu einem übergeordneten zusammengefasst und verschiedene Reihenfolgen als Modi dargestellt werden. Die Berücksichtigung von Material und vorhandenen Lagerflächen bedarf zusätzlichen Überlegungen und einer genauen Spezifikation der Problemstellung.

Die Modellierung von Bauprozessen wird aktuell mit verschiedenen Methoden testweise bearbeitet (vgl. Franz, 2010, S. 9):

- Erweiterten Petri-Netzen,
- Netzplantechnik,
- spezielle Simulationssprachen,
- bausteinorientierte Simulationswerkzeuge,
- agentenbasierte Simulationsumwicklungsumgebungen.

Im Kapitel 1.3 konnten diese Beschreibungssprachen auch bei anderen Ansätzen für die Modellierung von Produktionsprozessen klassifiziert werden. Die Ausarbeitungen dieser Arbeit deuten darauf hin, dass sich SysML besser zur Darstellung komplexer Prozesse eignet. Daher kann angenommen werden, dass das sich vorgestellte Modellierungskonzept und SysML zur Darstellung von Unikatprozessen eignen.

8.3.2 Ein praktisches Beispiel

Um die Modellierung im Bauwesen besser verstehen zu können, wurden Beispiele von Terminplänen betrachtet. Das Institut für Baubetriebswesen der Technischen Universität Dresden stellte für diese Arbeit einige Beispiele zur Verfügung. Tabelle 8-14 und Abbildung 8-10 zeigen einen sogenannten Terminplan für den Ausbau einer Büroanlage. Die Tabelle zeigt die benötigten Prozesse (Vorgangsname), deren Dauer, Vorgänger und Ressourcentypen (Ressourcenname) mit zugehöriger Gesamtarbeitszeit. In der zugehörigen Abbildung 8-10 werden die Vorgängerbeziehungen zur besseren Übersicht graphisch dargestellt.

Dies ist ein im Bauwesen typischerweise verwendeter Terminplan, anhand dessen die Projekte geplant und durchgeführt werden. Bei größeren Projekten ist es möglich, die Gesamtarbeitszeiten der Ressourcen anhand von Mengenzuteilung zu regulieren, so dass ein Arbeitsgang mit einer Ressource A beispielsweise 40 Stunden dauert und mit zwei Ressourcen A 25 Stunden. Zudem können die Terminpläne durch die vorhergehend beschriebenen Restriktionen ergänzt werden. Der gezeigte einfache Terminplan lässt sich dem Modellierungskonzept dieser Arbeit ohne Probleme unterordnen. Und auch die zuvor besprochenen Restriktion scheinen größten Teils vom Konzept vorgefertigt oder ergänzbar zu sein. Die ersten Betrachtungen sind also positiv zu bewerten.

Tabelle 8-14: Terminplan für den Ausbau einer Büroanlage

Nr.		Vorgangsname	Dauer	Anfang	Ende	Vorgänger	Ressourcennamen	Arbeit
1								
2		Fenstermontage (inkl. Fensterbrett)	5 Tage	27. Jul '10	02. Aug '10		Tischler	40 Std.
3		Jalousiemontage	5 Tage	27. Jul '10	02. Aug '10	2AA	Tischler	40 Std.
4								
5		Trockenbau (Ständerwerk)	4 Tage	30. Jul '10	04. Aug '10	2EA-2 Tage	Trockenbauer	32 Std.
6		Elektroinstallation (Grob)	3 Tage	03. Aug '10	05. Aug '10	5EA-2 Tage;2	Elektriker	24 Std.
7		Heizung (Grob)	4 Tage	03. Aug '10	06. Aug '10	5EA-2 Tage;2	Heizungsbauer	32 Std.
8		Klimatechnik (Grob)	3 Tage	03. Aug '10	05. Aug '10	5EA-2 Tage;2	Klimatechniker	24 Std.
9		Trockenbau (Wand Schließen)	3 Tage	09. Aug '10	11. Aug '10	8;7;6	Trockenbauer	24 Std.
10		Malerarbeiten	4 Tage	12. Aug '10	17. Aug '10	9	Maler	32 Std.
11		Türmontage	3 Tage	18. Aug '10	20. Aug '10	10	Tischler	24 Std.
12		Elektroinstallation (Fein)	3 Tage	18. Aug '10	20. Aug '10	10	Elektriker	24 Std.
13		Heizung (Fein)	3 Tage	18. Aug '10	20. Aug '10	10	Heizungsbauer	24 Std.
14		Klimatechnik (Fein)	3 Tage	18. Aug '10	20. Aug '10	10	Klimatechniker	24 Std.
15		Fußbodenlegearbeiten	4 Tage	23. Aug '10	26. Aug '10	14;13;12;11	Fußbodenleger	32 Std.
16		Möbelmontage (Einbaumöbel)	4 Tage	27. Aug '10	01. Sep '10	15	Tischler	32 Std.
17		Endreinigung	1 Tag	02. Sep '10	02. Sep '10	16	Gebäudereiniger	8 Std.

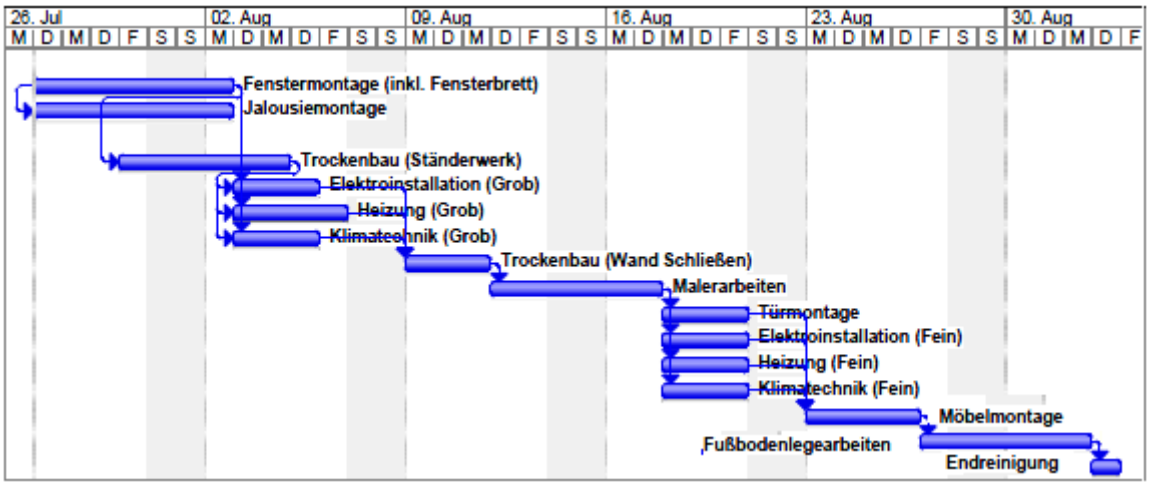


Abbildung 8-10: Terminplan für den Ausbau einer Büroanlage

8.4 Modellierung anderer Domänen

8.4.1 Modellierung in der Montage

Die Modellierung von Montageprozessen ist ein typisches Szenario im Untersuchungsgegenstand (vgl. Henlich et al., 2011; Rose et al., 2011). In einer Arbeit zeigte Gsuck, dass mit dem Metamodell dieser Arbeit ein Großteil komplexer Montagelinien abgebildet werden kann (vgl. Gsuck, 2009). Im Projekt *Manufacturing Resource-Optimization and Time-Scheduling Algorithm 2* (MARTA 2) wurde ein Metamodell für Montagelinien erstellt (vgl. Yang, 2010). In den folgenden Ausführungen werden Elemente und Eigenschaften erörtert, um die das Metamodell dieser Arbeit für die Abbildung von Montagelinien zu erweitern ist.

In der Montage gibt es Dienst- bzw. Pausenpläne (`Shiftplan` bzw. `Breakplan`). Ein Plan besteht aus einzelnen Schichten (`Shifts`) und Pausen (`Breaks`), die mit Ressourcen verbunden werden können. Hauptsächlich dienen sie für die Abbildung auf Menschliche Ressourcen. Eine Pause wie auch eine Schicht hat eine Startzeit (`startDate`) und eine Endzeit (`endDate`).

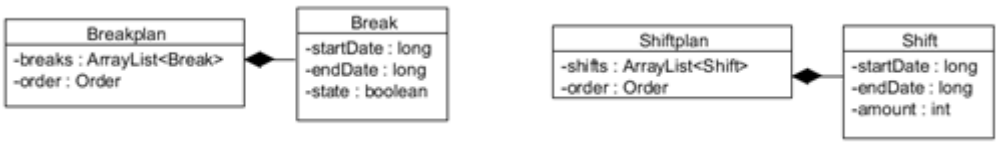


Abbildung 8-11: Die Stereotype *Breakplan* und *Shiftplan* (Yang, 2010, S. 6)

Eine weitere wichtige Erweiterung ist die Abbildung von Aufträgen (`Orders`), die aus einer Vielzahl von Produkten bestehen. Einem Auftrag werden somit Produkte zugeordnet. Produkte werden, wie im Modellierungskonzept dieser Arbeit üblich, durch das *Entity* und Aktivitätsdiagramme abgebildet.

Aufträge haben einen frühesten Startzeitpunkt (`earliestStart`) und einen frühsten Endzeitpunkt (`latestEnd`). Zudem haben sie eine eigene Priorität (`priority`) und bestehen aus verschiedenen Produkten, die ihnen als *Partpropertys* zugeordnet werden können.

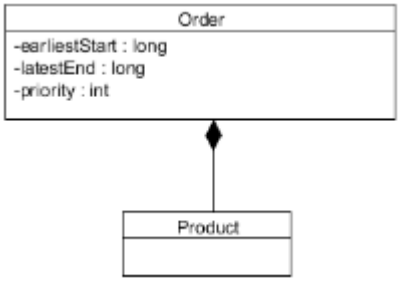


Abbildung 8-12: Das Stereotyp *Order* (Yang, 2010, S. 8)

8.4.2 Modellierung in der Transportlogistik

Auch in der Transportlogistik gibt es einen Bedarf für die Modellierung deskriptiver Systeme, was die durchgeführten Marktuntersuchungen von Simulationswerkzeugen aufzeigen. Über ein Drittel der betrachteten Simulatoren stellen Elemente für die Modellierung und Simulation der Transportlogistik bereit.

In einer zugrunde liegenden Arbeit analysierte Rehm logistische Modelle und ordnete sie in das Modellierungskonzept ein (vgl. Rehm/Schönherr, 2010). So ist es möglich, logistische Systeme als Netzwerkstrukturen mit Knoten und Kanten darzustellen. Als wesentliche Merkmale zeigen sich die Raumberücksichtigung (Transport), Zeitüberbrückungen (Lagern) und das Abändern der Anordnungen (Kommissionieren und Umschlagen). Die verschiedenen Ressourcen müssen diese Anforderungen umsetzen können. Erste Betrachtungen identifizieren Personen, Fördermittel und Lagertechnik als typische Ressourcen in der Logistik. Das *Entity* spezifiziert in der Logistik die im System befindlichen Sachgüter.

Die Ressourcen zum Transport werden in Stetig- und Unstetigförderer aufgeteilt. Unstetige Förderer benötigen eine Beschleunigung (`acceleration`), eine maximale Geschwindigkeit (`speed`), und eine maximale Ladekapazität (`capacity`). Stetige Fördermittel benötigen die Attribute Streckenlänge (`trackLength`), Förderabstand (`transferInterspace`) und mittlere Fördergeschwindigkeit (`averageSpeed`) spezifiziert. Zudem werden Lagerressourcen benötigt, bei denen die schon vorhandene Eigenschaft `capacity` jedoch genügt.

Die erweiterte Assoziation wird ebenfalls um Attribute erweitert. So sind in der Transportlogistik Freigabeanweisungen (`releaseDefinition`), Kommissionier-/Sortierregeln (`sortDefinition`) und Sicherheitsbestände (`inventoryLevel`) relevant. Abbildung 8-13 zeigt das für die Logistik erweiterte Metamodell.

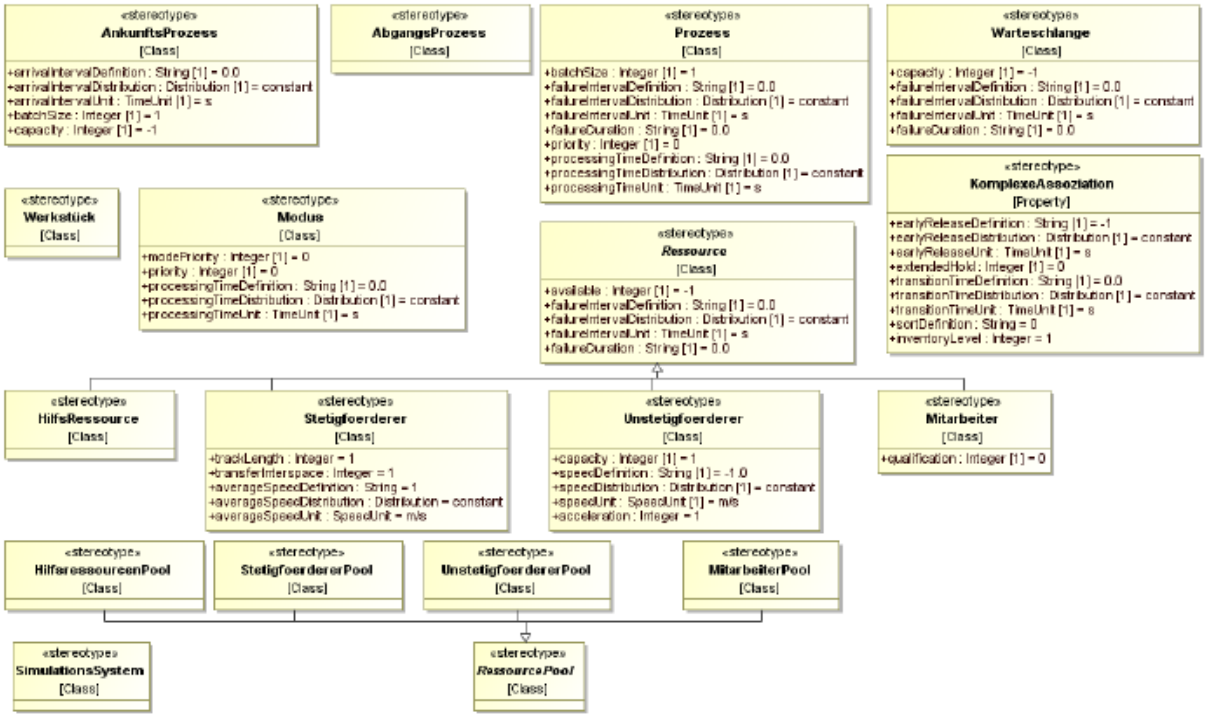


Abbildung 8-13: Erweitertes Metamodell für die Produktionslogistik (Rehm/Schönherr, 2010, S. 11)

9 Zusammenfassende Betrachtungen und Bewertung der Ergebnisse

Das Ziel der Dissertation war es, einen allgemeinen Ansatz für die Modellierung von diskreten Systemen aus dem Bereich der Produktion zu erarbeiten. Zudem sollte der Ansatz für die Nutzung vorbereitet werden. Um die Gesamtausgabe erfüllen zu können, wurden einleitend folgende Teilziele definiert:

1. Eine Systematik zur Klassifizierung und Beschreibung von Produktionsprozessen zu erarbeiten,
2. ein Konzept zur Abbildung der Systematik mit der Modellierungssprache SysML bereitzustellen,
3. ein Konzept zu evaluieren, mit dem allgemein modellierte Szenarien für eine möglichst breite Menge an Simulationswerkzeugen aufbereitet werden können, und die Entwicklung von entsprechenden Softwarewerkzeugen zur Übersetzung,
4. die Übersetzungen zu verifizieren und validieren,
5. ein geeignetes und freies Werkzeug zum Modellieren der Szenarien zu entwickeln,
6. ein nicht-proprietäres Simulationswerkzeug zu konzipieren,
7. angrenzende Domänen hinsichtlich einer möglichen Adoption des Konzeptes zu betrachten.

Abschließend sollen die Ergebnisse der Ausarbeitung nun zusammenfassend betrachtet und kritisch gewürdigt werden. Zudem erfolgt die Ausführung zukünftiger Potentiale und Anknüpfpunkte.

9.1 Eine Systematik zur Klassifizierung und Beschreibung von Produktionsprozessen.

Das erste Teilziel stellt den Schwerpunkt dieser Arbeit dar. Es sollte eine Systematik erarbeitet werden, mit der es möglich ist, Produktionsprozesse des abgegrenzten Untersuchungsgegenstandes zu beschreiben. Die erstellten Modelle sollten detailliert in ihre semantischen Teile zerlegt und abgegrenzt werden. Jede der semantischen Teileinheiten musste erörtert und konzeptionell erschlossen werden. Dabei galt die Anforderung, dass es möglich ist, eine große Menge der im Untersuchungsgegenstand vorkommenden Szenarien mit der Systematik zu beschreiben. Das Konzept unterscheidet die Rollen des Anwenders, der Produktionsszenarien modelliert, und des Entwicklers, der Simulatoren für den Untersuchungsgegenstand implementiert. Eine weitere Anforderung bestand darin, das Konzept strukturiert, flexibel, erweiterbar und vollkommen offengelegt zu gestalten.

In den betrachteten Modellierungskonzepten, wie derer verschiedener Simulationswerkzeuge, wurden die einzelnen semantischen Teile eines Modells meistens kaum differenziert. In der Literatur wird die Unterscheidung bisher oft nur zwischen Struktur und Verhalten durchgeführt. In der durchgeführten Literaturrecherche konnten nur Ansätze gefunden werden, aber kein Konzept zur deskriptiven Beschreibung des Untersuchungsgegenstandes. Zur Herleitung eines Konzeptes wurden neben der üblichen Literaturrecherche etliche Expertenbefragungen und zwei weitreichende Marktanalysen von Produktions- und Simulationswerkzeugen durchgeführt. Das dadurch entstandene Gesamtmodell, wurde in die neun Modellteile Experiment, Analyse, Struktur, Verhalten, Algorithmen, Steuerung, Zustand, Kommunikation und Ereignis unterteilt. Die Teilmodelle können partiell vom Anwender und vollständig vom Entwickler bearbeitet werden. Alle Teilmodelle konnten konzeptionell erschlossen werden. Die Anforderungen der Strukturiertheit, Flexibilität, Erweiterbarkeit und Offenlegung des Konzeptes wurden erfüllt.

Die Teilmodelle bilden einen Rahmen, der durch praktische Arbeiten mit dem Modellierungskonzept gefüllt werden kann. Mit dem Ansatz wurde zum ersten Mal eine umfassende Gliederung für die deskriptive Modellierung im Untersuchungsgegenstand gegeben. Die Gliederung besteht aus der Neunteilung des Gesamtkonzeptes und der Konzeption des jeweiligen Teilmodells. Es bietet sich an, die Gliederung für die Forschung, Lehre und praktische Arbeiten im Bereich der deskriptiven Modellierung diskreter Produktionsprozesse zur einheitlichen Verständigung zu nutzen. Die Betrachtung des erschlossenen Konzeptes zeigt, dass die Neunteilung des Gesamtmodells sinnvoll ist, da jedes der Modelle abgegrenzt und vielfältig in seiner Ausprägung ist.

Im achten Kapitel der Dissertation konnten erfolgreich erste Beispiele mit dem Modellierungskonzept dieser Arbeit abgebildet werden. Wünschenswerte Folgearbeiten dieser Dissertation beschäftigen sich mit der Umsetzung von weiteren praktischen Betrachtungen, um die Modellierungssystematik zu testen und zu verfeinern. Zudem wurde das Modellierungskonzept erfolgreich für Modelltransformationen mit den Simulatoren Flexsim, Simcron, Factory Explorer und Anylogic eingesetzt (vgl. Kapitel 4). Auch für die Entwicklung eines Simulationswerkzeuges wurde das Modellierungskonzept erfolgreich verwendet (vgl. Kapitel 7).

In wie weit das algorithmische und das Kontrollmodell umgesetzt werden können, so dass eine Einheit entsteht, die durch das Zusammenführen algorithmischer Aktionen Berechnungen durchführen kann, bleibt offen (vgl. Kapitel 3.5; vgl. Kapitel 3.6). Das experimentelle Modell sowie das Analysemodell sind in ihren Ansätzen beschrieben und können mit entsprechendem Aufwand weiter erschlossen werden.

9.2 Ein Konzept zur Abbildung der Systematik mit der Modellierungssprache SysML

Um die Systematik des Modellierungskonzeptes umsetzen zu können, war es notwendig:

1. Eine geeignete Modellierungssprache ausfindig zu machen,
2. sie auf ihre Anwendbarkeit auf die Konzeption der Modellierung hin zu überprüfen,
3. sie für die Anwendung des Konzeptes vorzubereiten.

Als geeignet wurde eine Modellierungssprache gewertet, wenn sie durch ein Gremium standardisiert, nicht proprietär, mächtig genug zur Beschreibung komplexer Sachverhalte, tauglich für deren (graphische) Darstellung und verwendbar zur Darstellung von domänenspezifischen Zielräumen ist. Da in der Dissertation mit den Modellen auch Modelltransformationen durchgeführt werden sollen, war es zudem notwendig, dass die Syntax der Sprache eindeutig definiert ist, um sie anwenden zu können.

Einführend wurden verschiedene Beschreibungssprachen auf ihre Eignung zum Abbilden des Modellierungskonzepts der Dissertation überprüft. Es wurden verschiedene Sprachen wie beispielsweise Modelica, Petri Netze, Moogo NG, unterschiedliche Graphen und Simulationssprachen begutachtet. Einzig SysML konnte die in der Dissertation gestellten Anforderungen erfüllen. Im zweiten Kapitel der Arbeit wurden die einzelnen Diagramme von SysML auf ihre Eignung für die Konzeption der Modellierung untersucht. Dabei war die Güte der Spezifikation, sowie die konzeptionelle Eignung ausschlaggebend (vgl. Kapitel 2.3.1; vgl. Kapitel 2.4.1). Das Blockdefinitionsdiagramm, Paketdiagramm, Aktivitätsdiagramm, Sequenzdiagramm und Zustandsdiagramm zeigten sich als geeignet. Um dem Leser der Arbeit die Ausführungen verständlich zu machen, zeigte das zweite Kapitel die Funktionsweise der als geeignet befundenen SysML-Diagramme auf. Im dritten Kapitel wurde parallel zur Konzeption der Modellierung deren Abbildbarkeit mit SysML erarbeitet. Die Systematik zur Beschreibung von Produktionsprozessen konnte vollständig mit SysML-Diagrammen abgebildet werden, somit war es möglich, das zweite Teilziel vollständig umzusetzen.

9.3 Ein Ansatz zur Transformation von Produktionsszenarien für Simulationswerkzeuge

Um die modellierten Szenarien nutzen zu können, ist es notwendig, sie in den Aufgabenbereich der für die Produktion üblichen Werkzeuge wie Simulatoren oder *Scheduler* zu überführen. Innerhalb der Dissertation war es daher notwendig, einen Ansatz zur Transformation der Szenarien zu entwerfen. Für diesen Zweck war es erforderlich:

1. Die Modelltransformationen dieser Arbeit zu klassifizieren um,

2. geeignete Methoden aus der Literatur zu begutachten um,
3. eine geeignete Methode auszuwählen um,
4. Transformationen an praktischen Beispielen durchzuführen.

Die Modelltransformationen dieser Arbeit wurden als exogene, horizontale Model-to-Model Transformationen klassifiziert, die mit operationalen und deklarativen Transformationsansätzen überführt werden können. Des Weiteren wurden verschiedene Methoden der Modelltransformation betrachtet, wie die von der OMG standardisierte QVT, imperative Modelltransformationen, deklarative Transformationen durch Graphgrammatiken, ATL, oAW und XSLT. Alle Ansätze, außer des imperativen, konnten aus verschiedenen Gründen nicht als geeignet befunden werden. Jedoch ist zu beachten, dass die Arbeiten am Transformationssystem im Jahr 2009 begannen, als die vielversprechende QVT noch in den Anfängen ihrer Entwicklung war. Auch heute scheint die QVT noch nicht ausreichend umgesetzt zu sein, jedoch könnte sie in naher Zukunft eine interessante Alternative darstellen.

Der imperative Ansatz wurde in ein mehrstufiges Transformationssystem integriert. Es wurden Transformationen von SysML nach den Simulatoren Anylogic, Flexsim, Factory Explorer und Simcron durchgeführt. Des Weiteren erfolgten Transformationen von Flexsim nach SysML .

Die verschiedenen Modelltransformationen konnten alle ohne größere semantische Schwierigkeiten durchgeführt werden, was die Flexibilität des Metamodells der Dissertation bekräftigt. Jedoch hat der imperative Transformationsansatz, der zu den operativen Ansätzen gehört, auch eine Kehrseite. Im Gegensatz zu den deklarativen Ansätzen ist er weniger strukturiert, was zu Nachteilen bei der Verifikation und Validierung führt. In einer Anschlussarbeit könnte die QVT bei ausreichendem Entwicklungsstand für die Modelltransformationen getestet werden.

9.4 Verifikation und Validierung der Übersetzungen

Um die unter dem dritten Teilziel beschriebenen transformierten Modelle auch nutzen zu können, ist es notwendig, sie zu verifizieren und zu validieren. Ansonsten besteht keine Aussagekraft darüber, in wie weit das Zielmodell dem Ausgangsmodell entspricht. Daher war es notwendig, für das Gesamtvorhaben ein Konzept zur Verifikation und Validierungen der Modelltransformationen bereitzustellen. Um dies zu erreichen war es erforderlich:

1. Kriterien herzuleiten, die aussagen wann eine Modelltransformation korrekt ist,
2. Zu untersuchen, an welchen Stellen innerhalb des Transformationssystems die Verifikation und Validierung unabdingbar ist,
3. Methoden auf ihre Eignung zur Verifikation und Validierung der Modelltransformationen zu begutachten,

4. die Methoden an die konkrete Problemstellung und an die Korrektheitskriterien anzupassen.

Als Kriterien für die Korrektheit einer Transformation wurden die syntaktische Korrektheit, syntaktische Vollständigkeit, semantische Korrektheit, Terminierung und Konfluenz gewählt. Beim Transformationssystem war es möglich, bei der Modellerstellung, während der Transformation vom Modellierungswerkzeug in das interne Modell und während der Transformation vom internen Modell zum Simulationsprogramm zu verifizieren und zu validieren. Da die anderen Stellen als weniger kritisch beurteilt wurden, erfolgte nur für den Transformationsschritt vom internen Modell zum Simulationswerkzeug eine detaillierte Verifizierung und Validierung.

Weiterhin wurden verschiedene Methoden für die Verifikation und Validierung auf Anwendbarkeit für die Modelltransformationen der Dissertation getestet. So erfolgte die Untersuchung von Formaler Verifikation durch *Modelchecking*, Formale Verifikation durch Theorem-Beweiser, Verifikation durch Korrespondenzkriterien, Verifikation durch Abhängigkeitsgraphen, Validierung durch Testen sowie Verifikation und Validierung durch Invarianten. Einige der Methoden prüften die Transformationen auf dem Metamodelllevel, andere auf Modelllevel. Da sich die anderen Methoden als ungeeignet herausstellten, wurden die Korrespondenzkriterien für die Überprüfung der syntaktischen Korrektheit, syntaktischen Vollständigkeit und semantischen Korrektheit auf Modelllevel gewählt. Die Überprüfung auf Konfluenz und Terminierung konnte mittels verschiedener Theoreme und einer strukturellen Codeanalyse auf Metamodelllevel ermittelt werden.

Da die gewählte Transformationsmethode operativ statt deklarativ war, erhöhte sich der Arbeitsaufwand für verschiedene Ansätze der Verifikation und Validierung enorm. Zudem steigerten die semantisch und syntaktisch kaum definierten Zielmodelle den Schwierigkeitsgrad für die Umsetzung der Verifikation und Validierung um ein Vielfaches. Die Überprüfung mit Korrespondenzkriterien bietet sich gerade bei operativ umgesetzten Modelltransformationen an. Weil die Verifikation mit Korrespondenzkriterien eine Modelllevel-Analyse ist, hängt die Aussagekraft der Ergebnisse von den gewählten Kriterien und Testszenarien ab. Das Verfahren kann ohne erheblichen Mehraufwand auf andere Übersetzer übertragen werden, da die Korrespondenzkriterien und Testszenarien übernommen werden können. Die Aussagekraft der Überprüfung auf Konfluenz und Terminierung ist durch die Verwendung einer semiformalen Codeanalyse nicht valide. Bei einer Folgearbeit, die einen deklarativen Ansatz – wie die QVT bei fortgeschrittenen Entwicklungsstand - nutzt, können andere Methoden für die Verifikation und Validierung genutzt werden.

9.5 Ein geeignetes und freies Werkzeug zum Modellieren von Produktionsszenarien mit SysML

Um das Modellierungskonzept nutzen zu können, ist es notwendig, auf ein geeignetes Werkzeug für die Erstellung und Darstellung der Modelle zugreifen zu können. Daher war es erforderlich, dem Anwender innerhalb der Arbeit ein geeignetes Werkzeug zur Verfügung zu stellen. Dafür wurden die folgenden Arbeitsschritte durchgeführt:

1. Spezifikation von Anforderungen an ein Modellierungswerkzeug,
2. durchführen einer Marktanalyse zur Erörterung des Angebotes,
3. Auswahl eines Werkzeuges und Begutachtung seines Arbeitsstandes,
4. Überarbeitung des Modellierungswerkzeuges anhand der erörterten Anforderungen.

Als grundlegende Anforderungen eines Modellierungswerkzeuges wurden ein geeignetes Austauschformat, die Unterstützung der aktuelle SysML Version und eine intuitive Bedienbarkeit gestellt. Um passende Anforderungen an ein Modellierungswerkzeug für die Anwendergruppe zu evaluieren, wurde ein *Workshop* durchgeführt, in dem Informatiker und Ingenieure *Usability*-Tests ausführten. Dabei wurden vor allem die Vielfalt und Abstraktheit von SysML als schwierig ermittelt.

Bei der Marktanalyse wurden proprietäre wie auch nicht proprietäre Werkzeuge analysiert. Kein proprietäres *Tool* konnte den Anforderungen gerecht werden. Da proprietäre Werkzeuge nicht durch Programmierarbeiten abgeändert werden können, fiel die Entscheidung auf ein nicht proprietäres Modellierungswerkzeug. TOPCASED qualifizierte sich durch seinen Arbeitsstand und wurde für die folgenden Arbeitsschritte als Grundlage gewählt. Unter dem Namen TOPCASED *Engineer* wurde das Modellierungswerkzeug in mehreren Belegarbeiten den herausgearbeiteten Anforderungen angepasst. Die gestellten Anforderungen konnten vollständig umgesetzt werden. Das Werkzeug TOPCASED *Engineer* wird auf Anfrage von der Professur für Modellbildung und Simulation der Universität der Bundeswehr München zur freien Verfügung gestellt.

9.6 Konzeption eines freien Simulationswerkzeuges basierend auf dem Modellierungskonzept der Arbeit

Um es zu ermöglichen, das Modellierungskonzept auch ohne Lizenzgebühren für praktische Tätigkeiten anwenden zu können, wurde ein Konzept für ein freies Simulationswerkzeug erarbeitet. Zur Umsetzung waren folgende Schritte notwendig:

1. Erörtern einer geeigneten Systemarchitektur mit geeigneten Komponenten,

2. Umsetzen eines geeigneten Simulationsalgorithmus,
3. Integration des Modells in den Simulator.

Die Systemarchitektur sollte möglichst viele vorgefertigte Komponenten benutzen, um den Arbeitsaufwand gering zu halten. Die Architektur des Simulators basiert auf dem freien Simulatorenframework JAMES II der Universität Rostock. Das Modell auf dem der Simulator arbeitet, basiert auf dem Modellierungskonzept dieser Dissertation und kann mit TOPCASED *Engineer* in SysML erstellt werden. Der Simulator arbeitet auf dem internen Modell des Transformationssystems in leicht erweiterter Form, dadurch kann für die Übertragung von TOPCASED *Engineer* zum Modell des Simulators der schon fertig entwickelte *Parser* genutzt werden.

Durch die gewählte Systemarchitektur konnte der Aufwand für die Entwicklung des Simulators minimiert werden. Der Simulator steht in seinen Grundfunktionalitäten zur Verfügung und wird auf Anfrage von der Professur für Modellbildung und Simulation der Universität der Bundeswehr München zur freien Verfügung gestellt. Anschließende Arbeiten können den Entwicklungsstand des Simulators ausbauen, verifizieren und validieren.

9.7 Betrachtung angrenzender Domänen auf eine mögliche Adoption des Konzeptes

Im letzten Teil dieser Arbeit wurden verschiedene Domänen innerhalb des Untersuchungsgegenstandes auf ihre Abbildbarkeit mit dem Modellierungskonzept der Dissertation betrachtet. Diese Untersuchungen waren notwendig, um das Modellierungskonzept auf seine Tauglichkeit hin untersuchen zu können. Zudem konnte es so domänenspezifisch erweitert werden. Einige Domänen wie die Halbleiterfertigung, Inlinefertigung von Solarzellen, Montage oder Logistik werden traditionell im Kontext der Modellierung und Simulation betrachtet und ermöglichten es so, das Modellierungskonzept anhand durchgeführter Arbeiten auf seine Adaptierbarkeit zu testen. Betrachtungen der Domäne der Unikatprozesse sind sehr aktuell im Kontext der Modellierung und Simulation. Daher wurde die Unikatfertigung hauptsächlich auf ihren Anwendungsnutzen, aber auch auf ihre Adaptierbarkeit getestet.

Alle Domänen ließen sich mit dem Modellierungskonzept der Dissertation abbilden, auch wenn das Metamodell jeweils mit domänenspezifischen Elementen oder Attributen erweitert werden musste. In der Inlinefertigung von Solarzellen konnte sogar eine vollständige Fertigung nachmodelliert werden. Der Nutzen der Unikatfertigung im Kontext der Modellierung und Simulation ist vermutlich vielversprechend und auch die Adaption ins Modellierungskonzept der Dissertation scheint integrierbar. Das Modellierungskonzept der Dissertation zeigte sich daher praxistauglich, der Aufgabenbereich wachsend.

9.8 Zukünftiger Forschungsbedarf

Die Auswertung der einzelnen Kapitel zeigt, dass die gestellten Arbeitsziele der Dissertation umgesetzt werden konnten. Wissenschaftlicher Fortschritt bringt Anknüpfungspunkte hervor, auf die anschließende Arbeiten aufbauen können. Abschließend wird der aus dieser Dissertation entstandene Forschungsbedarf zusammengefasst. Die Arbeit zeigt eine neue semantisch differenzierte Sichtweise auf diskrete Simulatoren und Modelle im Bereich der Produktion und Logistik. Zum einen ist es wünschenswert, praktische Arbeiten in verschiedenen Domänen durchzuführen, um das Modellierungskonzept weiter zu prüfen und mit Inhalten auszufüllen. Zum anderen bleibt es offen, das Teilkonzept der Steuerung zu implementieren und zu testen.

Es wurden Modelltransformationen zu verschiedenen Simulationswerkzeugen durchgeführt. Für die Transformationen wurde ein imperativer operationaler Ansatz gewählt, der vor allem auf Metamodelllevel schwer zu verifizieren und zu validieren ist. Die vielversprechende QVT könnte zukünftig eine geeignete Alternative für Modelltransformationen sein und mit ihrem deklarativen Teil eine verbesserte Verifikation und Validierung der in dieser Arbeit hergeleiteten Kriterien ermöglichen. Weiterhin wäre es wünschenswert, die Validierung und Verifikation der im TOPCASED Engineer erstellten Modelle weiter auszubauen.

Besonders interessant ist die weitere Umsetzung des auf JAMES II basierenden freien Simulationswerkzeuges. Während das Modellierungswerkzeug TOPCASED Engineer den Anforderungen der Dissertation entspricht, bedarf das Simulationswerkzeug noch Arbeitsaufwand. Der Simulationsalgorithmus sollte optimiert und die statistische Aufbereitungen an TOPCASED Engineer oder Excel vorbereitet werden. Weiterhin bietet es sich an, das Teilkonzept der Steuerung im freien Simulationswerkzeug zu implementieren. So kann das Steuerkonzept auf seine Umsetzbarkeit und Performance getestet und das Simulationswerkzeug umfassend eingesetzt werden.

Abschließend ist es möglich, Simulationsprojekte mit dem Modellierungskonzept der Dissertation bei den betrachteten Domänen durchzuführen. Bei diesen kann das ganzheitliche freie Modellierungskonzept der Dissertation auf zeitliche und finanzielle Vorteile bei praktischen Projekten überprüft werden.

Literaturverzeichnis

Abramovici, M./ Schulte, S. (2004). *PLM, logische Fortsetzung der PDM-Ansätze oder Neuauflage des CIM-Debakels?* Verfügbar unter: http://www.itm.rub.de/index.php?option=com_search&searchword=cim [26.06.2013].

Adam, D. (1998). *Produktionsmanagement,* (9. überarbeitete Auflage). Wiesbaden: Gabler.
Alten, H.-W./ Djafari Naini, A./ Folkerts, M./ Schlosser, H./ Schlote, K.-H./ Wussing, H. (2003). *4000 Jahre Algebra – Geschichte, Kulturen, Menschen.*. Berlin/ Heidelberg: Springer.
Angelidis, E./ Naumann, A./ Rose, O. (2012). An Extended Critical Path Method for Complex Assembly Lines. In *Proceedings of the 2012 Industrial Engineering Research Conference*, edited by G. Lim and J.W. Herrmann. Norcross, Georgia: IIE Institute of Industrial Engineers.

Atego (2013). Verfügbar unter: http://www.atego.com [29.06.2013].

Balzert, H. (2005). *UML 2 kompakt Checklisten* (2. Auflage). München: Spektrum.

Banks, J. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice.* Canada: Engineering and Management Press.

Barton, D. (2010). *An Experiment in Perfection. London*: BiblioBazar.

Baundry, B./ Ghosh, S./ Fleurey, F./ France, R./ Traon, Y./ Mottu, J.-M. (2009). Barriers to systematic model transformation testing. *Communication of the ACM*, 53, 139–143.

Becker, J. (1996). *Handelsinformationssysteme.* Landsberg am Lech: Moderne Industrie.

Beißert, U. (2010). Constraint-basierte Simulation von Bauprozessen – Studie zur Verwendung von Variablenordnungsstrategien. *Schriften der Professur Baubetrieb und Bauverfahren*, 19, 49-59.

Biehl, M. (2010). *Literature Study on Model Transformations*. Forschungsbericht des Royal Institute of Technology Stockholm.

Bloech, J./ Bogaschewsky, R./ Götze, U./ Roland, F./ Daub, A./ Buscher, U. (2003). *Einführung in die Produktion.* 5. überarbeitete Auflage. Berlin, Heidelberg, New York: Springer.

Bresser, T. (2004). Validierung und Verifikation (inkl. Testen, Model-Checking und Theorem Proving Analyse, Entwurf und Implementierung zuverlässiger Software.) unveröffentlichte Seminararbeit. Paderborn: Universität Paderborn, Fakultät Informatik.

Brüggemann, D. (2010). *Ein parametrisierbares Verfahren zur Änderungsplanung für den Flexible Flow Shop mit integrierter Schichtmodellauswahl.* Dissertation. Paderborn: Universität Paderborn, Fakultät Wirtschaftswissenschaften.

Cabot, J./ Clariso, R./ Guerra, E./ De Lara, J. (2010). Verification and validation of declarative model-to-model transformations through invariants. *The Journal of Systems and Software,* 83, 283–302.

Chaitin, G.J. (2007). *Thinking about Gödel and Turing, Essays on Complexity 1970 – 2007*. London: World Scientific.

Cleff, T. (2010). *Basiswissen Testen von Software: Vorbereitung zum Certified Tester (Foundation Level) nach ISTQB-Standard.* Witten: W3L.

Curry, G.L./ Feldman, R.M. (2011). *Manufacturing Systems Modeling and Analysis*. Heidelberg: Springer Verlag.

Czarnecki, K./ Helsen, S. (2003). Classification of Model Transformation Approaches. In 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Verfügbar unter: *http://www.softmetaware.com/oopsla2003/czarnecki.pdf* [27.06.2013].

Daub, A. (1994). *Ablaufplanung: Modellbildung, Kapazitätsabstimmung. und Unsicherheit*. Bergisch Gladbach, Köln: Eul.

Desrochers, A.A./ Al-Jaar, R.Y. (1995). *Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis*. Singapore: IEEE Press.

Dickmann, P. (2009). *Schlanker Materialfluss mit Lean Production, Kanban und Innovationen*. (2. Aktualisierte und erweiterte Auflage). Berlin, Heidelberg: Springer.

Domschke, W./ Drexl, A. (2006). *Einführung in Operations Research*. (7. Auflage). Berlin, Heidelberg, New York: Springer.

Domschke, W./ Scholl, A. (2006). *Heuristische Verfahren, Arbeits- und Diskussionspapiere der Wirtschaftswissenschaftlichen Fakultät*. Jena: Universität Jena, Fakultät Wirtschaftswissenschaften.

Domschke, W./ Scholl, A. (1997). *Produktionsplanung - Ablauforganisatorische Aspekte* (2. Auflage). Berlin: Springer.

Ehrig, H./ Ehrig, K. (2006). Overview of Formal Concepts for Model Transformations Based on Typed Attributed Graph Transformation. *Electronic Notes in Theoretical Computer Science*, 152, 3-22.

Ehrig, H./ Ehrig, K./ de Lara, J./ Taentzer, G./ Varro, D./ Gayapay, S. (2005). Termination Criteria for Model Transformation. In: Cerioli, M. (Hrsg.). *International Conference on Fundamental Approaches to Software Engineering*. Edinburgh, UK: Springer, 49–63.

Estler, H. (2009). Verifikation von Modelltransformationen. Unveröffentlichte Seminararbeit. Paderborn: Universität Paderborn, Fakultät Informatik.

Embedded Star (2013). Verfügbar unter: http://www.embeddedstar.com [29.06.2013].

Fachausschuss Software Engineering. (2004): Code Generierung und modellbasierte Softwareentwicklung für Luft- und Raumfahrtsysteme. Verfügbar unter: *http://www.t6.dglr.de/Veranstaltungen/2004_MDSE/DGLR_T64_2004_bericht.html* [19.05.2013].

Fleming, C./ Wach, M. (2010). Baubetriebliche Anforderungen an die diskret ereignisorientierte Modellierung und Simulation von Prozessen zur Planung und Fertigung von Unikaten. *Schriften der Professur Baubetrieb und Bauverfahren*, 19, 119-128.

Fowler M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd edition). Boston: Pearson Education.

Fowler, J.W./ Rose O. (2004). Grand challenges in modeling and simulation of complex manufacturing systems. *Simulation 2004*, 80, 469–476.

Frank, E. (2010). *Anpassung eines auf TOPCASED basierenden Modellierungswerkzeuges zur Abbildung kontinuierlicher Produktionsprozesse mit SysML*. Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Franz, V. (2010). Unikatprozesse und ASIM-Aktivitäten – Bericht von der Arbeitsgruppe „Unikatprozesse“. *Schriften der Professur Baubetrieb und Bauverfahren*, 19, 5-17.

Franz, V. (2011). Simulation von Unikatprozessen – Neue Anwendungen aus Forschung und Praxis. Verfügbar unter: *http://www.uni-kassel.de/upress/online/frei/978-3-86219-096-6.volltext.frei.pdf* [29.06.2013].

Fritzon, P. (2011). *Introduction to Modeling and Simulation of Technical and Physical Systems*. Singapore: IEEE PRESS.

Gaitanides, M. (1983). *Prozessorganisation. Entwicklung, Ansätze und Programme prozeßorientierter Organisationsgestaltung.* München: Vahlen.

GfSE Gesellschaft für Systems Engineering e.V. (2008): Tag des Systems Engineering. Verfügbar unter:
http://se-zert.com/index2.php?option=com_docman&task=doc_view&gid=9&Itemid=132
[19.05.2013].

Girault, C./ Rüdiger, V. (2003). *Petri nets for systems engineering - a guide to modeling, verification, and applications*. Berlin/ Heidelberg: Springer.

Glaser, H./ Geiger, W./ Rohde, V. (1992). *PPS- Produktionsplanung und-steuerung: Grundlagen, Konzepte, Anwendungen (*2. Auflage). Wiesbaden: Gabler.

Greenyer, J./ Rieke, J./ Travkin, O./ Kindler, E. (2008) TGGs for Transforming UML to CSP In. Contribution to the ACTIVE 2007 Graph Transformation Tools Contest. Verfügbar unter: *http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/tr-ri-08-287.pdf* [27.06.2013].

Grunske, L./ Geiger, L./ Lawley, M. (2007). A Graphical Specification of ModelTransformations with Triple Graph Grammars. Verfügbar unter: http://www.se.eecs.uni-kassel.de/fileadmin/se/courses/WebSeminarSS08/ExamplePaper.pdf [26.06.2013].

Gsuck, J. (2009). *Analyse der SysML als visuelle Dateneingabemethode im Vergleich zu MoogoNG und Implementierung eines Plugins für MARTA III*. Unveröffentlichte Diplomarbeit, Technische Universität Dresden, Fakultät Informatik.

Gutenberg, E. (1983). *Die Produktion, Grundlagen der Betriebswirtschaftslehre* (24. Auflage). Berlin, Heidelberg: Springer.

Hansmann, K.W. (2006) *Industrielles Management* (8. Auflage). München: Oldenbourg.

Harold, E.R. (2003) An Introduction to StAX. Verfügbar unter: *http://www.xml.com/pub/a/2003/09/17/stax.html* [27.06.2013].

Hause, M. (2006). The SysML Modelling Language. Verfügbar unter : *http://www.omgsysml.org/The_SysML_Modelling_Language.pdf* [25.06.2013].

Heckel, R./ Küster, J./ Taentzer, G. (2002). Confluence of Typed Attributed Graph Transformation Systems. In: Corradini, A./ Ehrig, H./ Kreowski, H./ Rozenberg, G. (Hrsg.). *Graph Transformations*. Barcelona: Springer, 20-36.

Henlich, T./ Weigert, G./ Klemmt, A. (2011). Modellierung und Optimierung von Montageprozessen. In: März, L./ Krug, W./ Rose, O./ Weigert, G. (Hrsg.). *Simulation und Optimierung in Produktion und Logistik.Praxisorientierter Leitfaden mit Fallbeispielen*. Berlin: Springer, 49-65.

Hermann, F./ Ehrig, H./ Orejas, F./ Golas, U. (2010). Formal analysis of functional behaviour for model transformations based on triple graph grammars. In: *Proceedings of the 5th international conference on Graph transformations*. Berlin/ Heidelberg: Springer, 155–170.

Hermann, F./ Hülsbusch, M./ König, B./ (2010). Specification and Verification of Model Transformations. *Electronic Communications of the EASST*, 30, 1-20.

Himmelspach, J./ Uhrmacher, A.M. (2009). The JAMES II framework for modeling and simulation. In: *Proceedings of the 2009 International Workshop on High Performance Computational Systems Biology*, Trento: IEEE Press, 101-102.

Huang, E./ Ramamurthy, R./ McGinnis, L. (2007). System and simulation modeling using SYSML. In: S.G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J.D. Tew,/ R.R. Barton (Hrsg.). *Proceedings of the 2007 Winter Simulation Conference*. Washington: IEEE Press, 796-802.

IBM (2013). Verfügbar unter: http://www.ibm.com [29.06.2013].

JAMES II (2013). Verfügbar unter: http://wwwmosi.informatik.uni-rostock.de/mosi/projects/cosa/james-ii/ [29.06.2013].

Jensen, K /Kristensen, L.M. (2009). *Coloured Petri Nets.* Berlin: Springer.

Jodlbauer, M. (2008). *Produktionsoptimierung, Wertschaffende sowie kundenorientierte Planung und Steuerung*. (2. Auflage). Wien, New York: Springer.

Jouault, F./ Allilaire, F./ Bézivin, J./ Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72, 31–39.

Kiener, S./ Maier-Scheubeck, N./ Obermaier, R./ Weiß, M. (2009): *Produktionsmanagement, Grundlagen der Produktionsplanung und –steuerung*. München: Oldenbourg.

Kindler, E./ Wagner, R.(2007). *Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios.* Paderborn: Universität Paderborn.

Kleijnen, J.P.C. (2008). *Design and Analysis of Simulation Experiments*. Berlin/ Heidelberg/ New York: Springer.

Kleppe, A.G./ Warmer, J./ Bast, W. (2003). *MDA Explained, the Model Driven Architecture: Practice and Promise*. Boston: Addison-Wesley Longman.

Kopte, T. (2012). In-line Solarzellen-Fertigung auf Siliziumbasis. Verfügbar unter: *http://www.solarfabrik2020.de/de/Projekte/In-line_Solarzellen.html* [29.06.2013].

Krög, M. (2011). Analytische Beschreibung und semiformale Abbildung von Verfahren und Algorithmen der Produktionssteuerung. Unveröffentlichte Masterarbeit, Technische Universität, Fakultät Betriebswirtschaftslehre.

Kurbel, K. (1999). *Produktionsplanung und -steuerung. Methodische Grundlagen von PPS-Systemen und Erweiterungen* (4. Auflage). München: Oldenbourg.

Kurbel, K. (2005). *Produktionsplanung und –steuerung im Enterprise Resource Planning und Supply Chain Management*. (6. Auflage). München: Oldenbourg.

Kurt, B. (2004). Computersimulationen *Physik Journal,* 3, 25-30.

Küster, J.M. (2004). Systematic Validation of Model Transformations. In IBM Zurich Research Laboratory. Verfügbar unter: *http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/Systematic_Validation_Model_Transformations.pdf* [27.06.2013].

Küster, J.M./ Abdelrazik, M. (2006). Validation of Model Transformations –First Experiences using a White Box Approach. In Proceedings Of MoDeVa (3rd International Workshop on Model Development, Validation and Verification). Verfügbar unter: *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.9126* [27.06.2013].

Küster, J.M./ Heckel, R./ Engels, G. (2006). Defining and validating transformations of UML Models. University of Paderborn. Verfügbar unter*: http://is.uni-paderborn.de/uploads/tx_sibibtex/Defining_and_Validating_Transformations_of_UML_Models.pdf* [27.06.2013].

Lange, T. (2008). *Conceptual Modeling of Simulation Models Using SysML*. Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Lano, K./ Kolahdouz, S. (2010). Specification and verification of model transformations using UML-RSDS. In *Proceedings of the 8th international conference on Integrated formal methods*. Berlin/ Heidelberg: Springer, 20-36.

Lara, J./ Taentzer, G. (2004). Gabriele: Automated Model Transformation and Its Validation Using AToM 3 and AGG. In Blackwell, A. F./ Marriott, K./ Shimojima, A. (Hrsg.). *Diagrammatic Representation and Inference*. Stanford, CA: Springer, 182-198.

Law, A.M./ Kelton, W.D. (2000). *Simulation Modelling and Analysis.* Boston: McGraw-Hill.

Leitner, J. (2006). *Verifikation von Modelltransformationen basierend auf Triple Graph Grammatiken.* Unveröffentlichte Diplomarbeit, Technische Universität Berlin, Fakultät Informatik.

Lindemann, M./ Schmidt, S. (2007) Marktübersicht: in Produktion und Logistik. In: *PPS-Management: Zeitschrift für ERP-Systeme in Produktion und Logistik*, Vol. 12.2007, S.48-55

Lißke, D. (2009). Entwurf eines Modellierungstools für SysML als Eclipse Plug-In. Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Little, J.D.C. (1961). A Proof of the Queueing Formula L = λ W. *Operations Research,* 9, 383–387.

Lödding, H. (1998): *Verfahren der Fertigungssteuerung*. Berlin/ Heidelberg/ New York: Springer.
Majohr, M. (2008). *Heuristik zur personalorientierten Steuerung von komplexen Montagesystemen.* Dresden: TUDpress.

Mathar, H.-J., Scheuring, J. (2009): *Unternehmenslogistik, Grundlagen für die betriebliche Praxis mit zahlreichen Beispielen, Repetitionsfragen und Antworten.* Compendio: Zürich.

Mangano, S. (2006). *XSLT Kochbuch.* Köln: O'Reilly.

März, L./ Krug, W./ Rose, O./ Weigert G. (2011). *Simulation und Optimierung in Produktion und Logistik.Praxisorientierter Leitfaden mit Fallbeispielen*. Berlin: Springer.

Mens, T. (2010). Model Transformation: A Survey of the State-of-the-Art. In: Babau, J.-P./ Fornarino, M./ Champeau, J./ Robert, S. (Hrsg.), *Model Driven Engineering for distributed Real-Time Systems* (S. 1-19). Published Online: Wiley-VCH.

Mens, T./ Van Gorp, P. (2005). A taxonomy of model transformation. In Proc. Int'lWorkshop on Graph and Model Transformation. Verfügbar unter: *http://drops.dagstuhl.de/volltexte/2005/11/pdf/04101.SWM2.Paper.pdf* [27.06.2013].

Miller, J./ Murkerji, J. (2003). MDA Guide Version 1. Verfügbar unter: *http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf* [26.06.2013].

Miller, M./ Kondruweit, F. (2012). Prozesse für die kostengünstigere Produktion von Solarzellen. Verfügbar unter: *http://www.ist.fraunhofer.de/content/dam/ist/de/documents/pi57_pvsec.pdf* [29.06.2013].

Montevechi, J.A.B./ Miranda, R./ Friend, J.D. (2009). Sensitivity Analysis in Discrete-Event Simulation Using Design of Experiments. Verfügbar unter: *http://cdn.intechopen.com/pdfs/38819/InTech-Sensitivity_analysis_in_discrete_event_simulation_using_design_of_experiments.pdf* [26.06.2013].

Moss, H. (2012). *TOPCASED Engineer ein modernes SysML Modellierungswerkzeug für Ingenieure*. Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Modelica (2013). *Modelica*. Verfügbar unter: https://www.modelica.org/ [26.06.2013].

Mudo, S. (2011). *Entwicklung eines Flexsim Übersetzer-Plugins*. Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Narayana, A./ Karsai, G. (2008). Verifying Model Transformations by Structural Correspondence. In Electronic Community of European Association of Software Science and Technology. Verfügbar unter: *http://journal.ub.tu-berlin.de/eceasst/article/view/157* [27.06.2013].

Neumann, K. / Schwindt, C. (1997). Activity-on-node networks with minimal and maximal time lagsand their application to make-to-order production. *Spectrum*, 19, 205-217.

Nipkow, T./ Lawrence, C./ Paulson, M./Wenzel, M. (2002) *Isabelle/HOL A Proof Assistant for Higher-Order Logic.* Verfügbar unter: http://isabelle.in.tum.de/doc/tutorial.pdf. [29.06.2013].

Nguyen, K.D./ Thiagarajan, P.S./ Wong, W.-F. (2007). A UML-based Design Framework for Time-triggered Applications. Verfügbar unter: *http://www.comp.nus.edu.sg/~thiagu/public_papers/rtss_es07.pdf* [15.06.2013].

No Magic (2013). Verfügbar unter: http://www.nomagic.com [29.06.2013].

Noack, D. (2012). Online Simulation in Semiconductor Manufacturing. Dissertation, Universität der Bundeswehr München. Verfügbar unter: *http://athene.bibl.unibw-muenchen.de:8081/doc/90413/90413.pdf* [27.06.2013].

Noche, B./ Wenzel, S. (2000*).* The new simulation in production and logistics. Prospects, views and attitudes. In: Mertins, K./ Rabe, M. (Hrsg.). *9. ASIM-Fachtagung Simulation in Produktion und Logistik* (S. 423-434). Berlin: Eigenverlag.

Nolte, S. (2009). *QVT - Relations Language*. Berlin/ Heidelberg/ New York: Springer.

Oestereich, B. (2006). *Analyse und Design mit UML 2.1*. München: Oldenbourg.

OMG (2009). Ontology Definition Metamodel. Verfügbar unter: *http://www.omg.org/spec/ODM/1.0/* [25.06.2013].

OMG (2010). SysML Modelling Language explained. Verfügbar unter: *http://www.omgsysml.org/SysML_Modelling_Language_explained-finance.pdf* [25.06.2013].

OMG (2012). OMG Systems Modeling Language Version 1.3. Verfügbar unter: *http://www.omg.org/spec/SysML/1.3/PDF/* [25.06.2013].

OMG (2013). MetaObject Facility. Verfügbar unter: *http://www.omg.org/mof/* [25.06.2013].

Osherove, R. (2010). *The Art of Unit Testing.* Heidelberg: Hüthig Jehle Rehm.

Pappert, F. (2007). *Erstellung eines Flexsim Plug-Ins.* Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Pappert, F. (2010). *Abbildung der Produktion von Inlinefertigung von Solarzellen.* Unveröffentlichte Diplomarbeit, Technische Universität Dresden, Fakultät Informatik.

Papyrus (2013). Verfügbar unter: http://www.papyrusuml.org [29.06.2013].

Partzsch, T. (2009). *Realization of a meta-model for simulating discrete processes in production with MagicDraw and Java.* Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Partzsch, T. (2010). *Modeling and parsing of SysML models in MagicDraw and TOPCASED.* Unveröffentlichte Diplomarbeit, Technische Universität Dresden, Fakultät Informatik.

Pawellek, G. (2007). *Produktionslogistik: Planung-Steuerung-Controlling.* München: Hanser.

Petri, C.A. (1962). *Kommunikation mit Automaten.* Bonn: Schriften des Institutes für instrumentelle Mathematik der Universität Bonn.

Pielok, T. (1995). *Prozesskettenmodulation – Management von Prozessketten mit Hilfe von Logistic Function Deployment.* Dortmund: Verlag Praxiswissen.

Pietsch, H. (2011). Simulation von Unikatprozessen Herausforderung für kommerzielle Simulationswerkzeuge. In: Franz, V. (Hrsg.). *Simulation von Unikatprozessen – Neue Anwendungen aus Forschung und Praxis*. Verfügbar unter: *http://www.uni-kassel.de/upress/online/frei/978-3-86219-096-6.volltext.frei.pdf* [29.06.2013].

Plump, D. (1998) Termination of graph rewriting is undecidable. *Fundamenta Informaticae,* 33, 40-50.

Pörnbacher, C. (2010). *Modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme.* München: Herbert Utz.

Potoradi, J./ Boon, O.S./ Mason, S.J. , Fowler, J.W./ Pfund, E.M. (2002). Using simulation-based scheduling to maximize demand fulfillment in a semiconductor assembly facility. In: Yücesan, E./ Chen, C.H./ Snowdon, J.L./ Charnes, J.M. (Hrsg.). *Proceedings of the 2002 Winter Simulation Conference.* Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Rabe, M./ Spiekermann, S./ Wenzel, S. (2006). *Verifikation und Validierung für die Produktion und Logistik*. Berlin/ Heidelberg: Springer.

Rausch, V. (2010). *Bediensysteme der Instandhaltung. Eine Verknüpfung von mathematisch-statistischen Methoden und der Bedientheorie.* Saarbrücken: Südwestdeutscher Verlag für Hochschulschriften.

Rehm, M. (2009). *Konzipierung eines allgemeingültigen Daten-Metamodells zur Beschreibung von Produktionssystemen als Grundlage für die automatische Simulationsmodellgenerierung.* Unveröffentlichte Masterarbeit, Technische Universität Dresden, Fakultät Betriebswirtschaftslehre.

Rehm, M./Schönherr, O./Schmidt, T. (2010). Ein Metamodell von Produktionssystemen als Grundlage für die automatische Simulationsmodellgenerierung. In: Overmeyer, Ludger (Hrsg.). *Tagungsband zum 6. Fachkolloquium der Wissenschaftlichen Gesellschaft für Technische Logistik (WGTL)* (S. 141-152). Berlin: TEWISS Verlag.

Reisig, W. (2010). *Petrinetze - Modellierung, Analyse, Fallstudien*. Wiesbaden: Vieweg-Teubner.

Rose, O./ Majohr, M./ Angelidis, E./ Pappert, F./ Noack, D. (2011). Personaleinsatz- und Ablaufplanung für komplexe Montagelinien mit MARTA 2. In: März, L./ Krug, W./ Rose, O./ Weigert, G. (Hrsg.). *Simulation und Optimierung in Produktion und Logistik.Praxisorientierter Leitfaden mit Fallbeispielen*. Berlin: Springer.

Rosenberg, D./ Mancerella, S. (2010). Embedded Systems Development using SysML: An Illustrated Example using Enterprise Architect. Verfügbar unter: *http://www.sparxsystems.com.au/downloads/ebooks/Embedded_Systems_Development_using_SysML.pdf* [25.06.2013].

Rücker, T. (2006). *Optimale Materialflusssteuerung in heterogenen Produktionssystemen*. (1. Auflage). Dissertation. Gabler, Wiesbaden: TU Ilmenau.

Rumbaugh, J./ Jacobson, I./ Booch, G. (2004). *The Unifield Modeling Language Reference Manual.* Boston: Addison-Wesley.

Rupprecht, C. (2002). *Ein Konzept zur projektspezifischen Individualisierung von Prozessmodellen.* Hochschulschrift Universität-Karlsruhe.

Russell R.B. (2010). Simulation Experiment Design. Verfügbar unter: *http://www.informs-sim.org/wsc10papers/009.pdf* [26.06.2013]

Sauer, W. (2003). *Prozesstechnologie der Elektrotechnik, Modellierung, Simulation und Optimierung der Fertigung.* München: Hanser.

SAX (2010). Der SAX Parser.Verfügbar unter: *http://www.saxproject.org* [29.06.2013].

Scharfe, D. (2011). *Entwicklung und Validierung eines Plugins zur Übersetzung kontinuierlicher Produktionsprozesse von SysML nach Factory Explorer.* Unveröffentlichte Masterarbeit, Technische Universität Dresden, Fakultät Informatik.

Schatten, A./ Biffl, S./ Demolsky, M./ Gostischa-Franta, E./ Streichert, T./ Winkler, D. (2010). *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen.* Heidelberg: Spektrum.

Scheer, A.-W. (1990). *Computer Integrated Manufacturing* (4. Auflage). Berlin: Springer.

Schmidt, G. (1997). *Prozess Management Modelle und Methoden*. Berlin/ Heidelberg: Springer.

Schneeweiß, C. (2002). *Einführung in die Produktionswirtschaft.* Berlin: Springer.

Schönherr, O. (2008). *Ein Allgemeines SysML Modell zur Abbildung diskreter Prozesse in der Produktion.* Unveröffentlichte Diplomarbeit, Technische Universität Dresden, Fakultät Informatik.

Schönherr, O./ Rose, O. (2009). A General SysML Model for Discrete Processes in Production Systems. In: Lee, L.H./ Kuhl, M.E./ Fowler, J.W./ S. Robinson (Hrsg.). *Proceedings of the 2009 INFORMS Simulation Society Research Workshop*. Coventry, UK: INFORMS Simulation Society, University of Warwick, 130-136.

Schönherr, O./ Rose, O. (2010). Important Components for Modeling Production Systems with SysML. Proceedings of the 2010 IIE Annual Conference and Expo (S. 364-376). Cancun, Mexico: IEEE Press.

Schulze, S. (2008). *Anwenderforum SysML*. Hamburg: Gesellschaft für Systems Engineering e.V.

Schuster, J. (2010). *Code Generierung für modellbasierte Testsysteme von Fahrzeugsteuergeräten.* Unveröffentlichte Bachelorarbeit, Technische Universität München, Fakultät Informatik.

SEMI E10., (2001). *Specification for Definition and Measurement of Equipment Reliability, Availability, and Main-tainability, Book of SEMI Standards.* Mountain View, CA: SEMI.

Seemann, J./ Gudenberg, J. (2000). *SoftwareEntwurf mit UML.* Berlin: Springer.

Seemann, J./ Gudenberg, J. (2006). *SoftwareEntwurf mit UML 2* (2. Auflage). Berlin: Springer.

Seidel, U.A. (1998). *Verfahren zur Generierung und Gestaltung von Montageablaufstrukturkomplexer Erzeugnisse. Berlin:* Springer.

Sharapov, M. (2010). *Development of a modeling tool for the representation of continuous production processes with SysML.* Unveröffentlichte Masterarbeit, Technische Universität Dresden, Fakultät Informatik.

Silva, M./ Teruel, E. (1997). Petri Nets for the design and operation of manufacturing systems. *European Journal of Control*, 3, 182–199.

Simcron GmbH (2003). Simcron Dokumentation. Verfügbar unter: *http://www.simcron.de* [27.06.2013].

Skulschus, M. (2012). *PHP - OOP, Design Patterns und UML*. Berlin: Comelio.

Sparx Systems (2013). Verfügbar unter: http://www.sparxsystems.com.au [29.06.2013].

Spearman, M.L./ Hopp, W.J. (2011). *Factory Physics.* Boston: Waveland Press.

Stachowiak, H. (1973).*Allgemeine Modelltheorie.* Berlin: Springer.

Störrle, H. (2005). *UML 2 für Studenten*. München: Pearson.

Störrle, H. (2004). Semantics of Control-Flow in UML 2.0 Activities. In: Bottoni, P./ Hundhausen, C./ Levialdi, S./ Tortora, G. (Hrsg.). *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*. Berlin: Springer. 235-242.

Störrle, H. (2005). *UML 2 für Studenten.* München: Pearson.

Sun Microsystem (2013). Java™ Platform, Standard Edition 6 Specification. Verfügbar unter: *http://docs.oracle.com/javase/6/docs/api/* [29.06.2013].

Tanenbaum, A. S. (2009). *Moderne Betriebssysteme.* München: Pearson.

Tempelmeier, H. (2005). *Produktion und Logistik.* Berlin: Springer.

TOPCASED (2013). Verfügbar unter: http://www.topcased.org [29.06.2013].

Varro, D./ Pataricza, A. (2003). Automated Formal Verification of Model Transformations. In Jürjens, J. , Rumpe, B./ France, R./ Fernandez, E.B. (Hrsg.) . *Critical Systems Development in UML. Proceedings of the UML'03 Workshop* (S. 63-78). Technical Reports, Technische Universität München.

VDI (1997). VDI Richtlinie 3633. Verfügbar unter: http://www.beuth.de/de/technische-regel-entwurf/vdi-3633/1199512 [26.06.2013].

Wannenwetsch, H. (2009): *Integrierte Materialwirtschaft und Logistik, Beschaffung, Logistik, Materialwirtschaft und Produktion.* (4. Aktualisierte Auflage). Springer: Heidelberg, Dordrecht, London, New York.

Weilkiens, T. (2006). *System Engineering mit SysML/UML*. Heidelberg: Dpunkt.

Weilkiens, T. (2006). *UML-2-Zertifizierung : Fundamental, Intermediate und Advanced*. *Test-Vorbereitung zum OMG Certified UML Professional*. Heidelberg: Dpunkt.

Weilkiens, T. (2008). *Systems Engineering with SysML/UML*. Heidelberg: Dpunkt.

Weilkiens, T./ Oestereich, B. (2006). *UML-2-Zertifizierung : Fundamental, Intermediate und Advanced*. *Test-Vorbereitung zum OMG Certified UML Professional*. Heidelberg: Dpunkt.

Wenzel, S. (2009). Modellbildung und Simulation in Produktion und Logistik – Stand und Perspektiven. In: Elst, G. (Hrsg.). *Tagungsband zum ASIM-Treffen STS/GMMS*. Stuttgart: Fraunhofer IRB Verlag, 7-16.

Winter, E. (2012). *Validierung und Verifikation der automatischen Modelltransformation von SysML nach Flexsim.* Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Winzker, M. (2007). *Elektronik für Entscheider: Grundwissen für Wirtschaft und Technik.* Wiesbaden: Vieweg Praxiswissen.

Wright Williams & Kelly (2014). *Factory Explorer*. Verfügbar unter: http://www.wwk.com/. [29.06.2013].

Yang, G. (2004). *Produktionsplanung in komplexen Wertschöpfungsnetzwerken, ein integrierter hierarchischer Ansatz in der chemischen Industrie.* Wiesbaden: Gabler.

Yang, Y. (2010) *Development of an Anylogic Plugin.* Unveröffentlichte Masterarbeit, Technische Universität Dresden, Fakultät Informatik.

Zhou, M./ Venkatesh, K. (1999). *Modelling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach.* Singapore: World Scientific Printers.

Zirakadze, G. (2010*). Entwicklung eines Translator-Plugins für den Simcron MODELLER 3.* Unveröffentlichte Belegarbeit, Technische Universität Dresden, Fakultät Informatik.

Aus der Dissertation entstandene Fachbeiträge

Schönherr, O./ Rose, O. (2009). Ein Allgemeines SysML Modell zur Abbildung diskreter Prozesse in der Produktion. *In Proceedings of the 2009 Tag des System Engineerings*. Hamburg.

Schönherr, O./ Rose, O. (2009). A General SysML Model for Discrete Processes in Production Systems. In L. H.Lee/ M. E. Kuhl/ J. W. Fowler/ S. Robinson (Hrsg.), *Proceedings of the 2009 INFORMS Simulation Society Research Workshop (*S. 130-136), Coventry, U.K.: INFORMS Simulation Society, University of Warwick.

Schönherr, O./ Rose, O. (2009). First Steps towards a general SysML model for discrete processes in production systems. In M. D. Rossetti/ R. R. Hill/ B. Johansson, A. Dunkin/ R. G. Ingalls (Hrsg.). *Proceedings of the 2009 Winter Simulation Conference* (S. 2206-2218). Austin, Texas: IEEE Press.

Schönherr, O./ Rose, O. (2010). Important Components for Modeling Production Systems with SysML. Proceedings of the 2010 IIE Annual Conference and Expo (S. 364-376). Cancun, Mexico: IEEE Press.

Schönherr, O./ Rose, O. (2010). Modelling of Production Systems with SysML. In G. Zülch/ P. Stock (Hrsg.). *Proceedings of the 2010 ASIM-Fachtagung Simulation in Produktion und Logistik* (S. 453-460). Karlsruhe: KIT Scientific Publishing.

Schönherr, O./ Rose, O. (2011). A general model description for discrete processes. In S. Jain, R.R. Creasey/ J. Himmelspach/ K.P. White/ M. Fu (Hrsg.). *Proceedings of the 2011 Winter Simulation Conference* (S. 2206-2218). Philadelphia: IEEE Press.

Schönherr, O., Moss/ J.H., Rehm/ M. & Rose, O. (2012). A free simulator for modeling production systems with SysML. In C. Laroque/ J. Himmelspach/ R. Pasupathy/ O. Rose/ A.M. Uhrmacher (Hrsg.). *Proceedings of the 2012 Winter Simulation Conference*. Berlin: IEEE Press.

Schönherr, O./ Pappert, F./ Rose, O. (2013). Domain Specific Simulation Modeling with SysML and Model-to-Model Transformation for Discrete Processes. In Pau Fonseca i Casas (Hrsg.). *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications* (S. 275-312). United States of America: IGI Global.

Rehm, M./ Schönherr, O./ Schmidt, T. (2010). Ein Metamodell von Produktionssystemen als Grundlage für die automatische Simulationsmodellgenerierung. In: Overmeyer, Ludger (Hrsg.). *Tagungsband zum 6. Fachkolloquium der Wissenschaftlichen Gesellschaft für Technische Logistik (WGTL)* (S. 141-152). Berlin: TEWISS Verlag.

Rehm, M./ Schönherr, O./ Schmidt, T. (2010). Ein Metamodell von Produktionssystemen als Grundlage für die automatische Simulationsmodellgenerierung. *Logistics Journal, Proceedings*, 06., 1-12.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADDL	*Architecture Analysis and Design Language*
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
APS	*Advanced Planning and Scheduling*
ASCET	*Advanced Simulation and Control Engineering Tool*
ASIM	Arbeitsgemeinschaft Simulation
ATL	*Atlas Transformation Language*
BDD	Blockdefinitionsdiagramm
bzw.	beziehungsweise
CIM	*Computer Integrated Manufacturing*
CDT	*C/C++ Development Tools*
CPL	*Common Public License*
CSP	*Constraint Satisfaction* Problem
DBF	Dezentralen Bestandsorientierten Fertigungsregelung
DES	*Discrete Event* Simulator
DIFN	Diffusion
DOM	*Document Object Model*
DOORS	*Dynamic Object Oriented Requirements System*
DOPG	*Doping*
EMF	*Eclipse Modeling Framework*
EPL	*Eclipse Public License*
Et al.	und andere
FCFS	*First Come First Served*
FIFO	*First In First Out*
GFSE	Gesellschaftfür Systems *Engineering*

GmbH	Gesellschaft mit beschränkter Haftung
HNDA	*Material Handler*
IEC	*International Electrotechnical Commission*
INCOSE	*International Council on Systems Engineering*
JAMES II	*Java based Multipurpose Environment for Simulation II*
LHS	*Left Hand Side*
M2M	*Modelt to Model*
MARTA	*Manufacturing Resource-Optimization and Time-Scheduling Algorithm*
MATLAB	MATrix LABoratory
MBSE	*Model Based Systems Engineering*
MDD	*Model Driven Development*
MMT	*Model-to-Model Transformation*
MOF	*Meta-Object-Facility*
MSL	Modelica Standard Library
oAW	*open Architecture Ware*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
PEVD	*Plasma Etch Chemical Vapor Deposition*
PGG	*Pair Graph Grammars*
PPS	Produktionsplanungs- und Steuerungssystem
QVT	*Query View Transformation*
RHS	*Right Hand Side*
RSDP	*Rational's Software Development Platform*
SAM	*Structured Analysis Modeling*
SAX	*Simple API for XML*
SCM	*Supply Chain Managements*
SORT	*Sorting*
StAX	*Streaming API for XML*

TEXT	*Texturing*
TGG	*Triple Graph Grammars*
TOPCASED	*Toolkit in Open Source for Critical Applications and Systems Development*
TU	Technische Universität
UML	*Unified Modeling Language*
VDI	Verein deutscher Ingenieure
vgl.	vergleich
VKN	Vorgangsknotennetzpläne
VPN	Vorgangspfeilnetzpläne
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformations*

Anhang

Evaluationsbogen zum Testen der Modellierungsqualität

1. Angaben zur Person

Name_

Erfahrungen mit der Modellierung von UML:

Erfahrungen mit der Modellierung von SysML:

Erfahrung mit der Modellierung mittels Simulationswerkzeugen:

2. Allgemeine Angaben

trifft voll zu *trifft gar nicht zu*

Das Konzept von SysML ist verständlich.

☐ ☐ ☐ ☐ ☐ ☐

Die strukturelle Modellierung mit SysML ist verständlich.

☐ ☐ ☐ ☐ ☐ ☐

Die Modellierung des Verhaltens mit SysML ist verständlich.

☐ ☐ ☐ ☐ ☐ ☐

3. Angaben zu Szenario 1

trifft voll zu *trifft gar nicht zu*

Die Aufgabenstellung konnte komfortabel modelliert werden.

☐ ☐ ☐ ☐ ☐ ☐

Ich konnte alle Elemente, die ich zum Modellieren benötigte, finden.

☐ ☐ ☐ ☐ ☐ ☐

Ich konnte die zum modellieren benötigten Elemente ohne größeren Aufwand finden.

☐ ☐ ☐ ☐ ☐ ☐

Die Oberfläche von MagicDraw war übersichtlich.

☐ ☐ ☐ ☐ ☐ ☐

SysML war zum Beschreiben des Szenarios geeignet.

☐ ☐ ☐ ☐ ☐ ☐

Benötigte Zeit zum Modellieren des Szenarios: __ min

Anmerkungen zur Verbesserung der Usability

_____

_____

_____

_____

_____

Allgemeine Anmerkungen

_____

_____

_____

_____

_____

4. Angaben zu Szenario 2

trifft voll zu *trifft gar nicht zu*

Die Aufgabenstellung konnte komfortabel modelliert werden.

☐ ☐ ☐ ☐ ☐ ☐

Ich konnte alle Elemente, die ich zum Modellieren benötigte, finden.

☐ ☐ ☐ ☐ ☐ ☐

Ich konnte die zum Modellieren benötigten Elemente ohne größeren Aufwand finden.

☐ ☐ ☐ ☐ ☐ ☐

Die Oberfläche von MagicDraw war übersichtlich.

☐ ☐ ☐ ☐ ☐ ☐

SysML war zum Beschreiben des Szenarios geeignet.

☐ ☐ ☐ ☐ ☐ ☐

Die Modellierung des Szenarios gelang einfacher als beim Szenario 1.

☐ ☐ ☐ ☐ ☐ ☐

Benötigte Zeit zum Modellieren des Szenarios: __min

Anmerkungen zur Verbesserung der Usability

_____

_____

_____

_____

_____

Allgemeine Anmerkungen

_____

_____

_____

_____