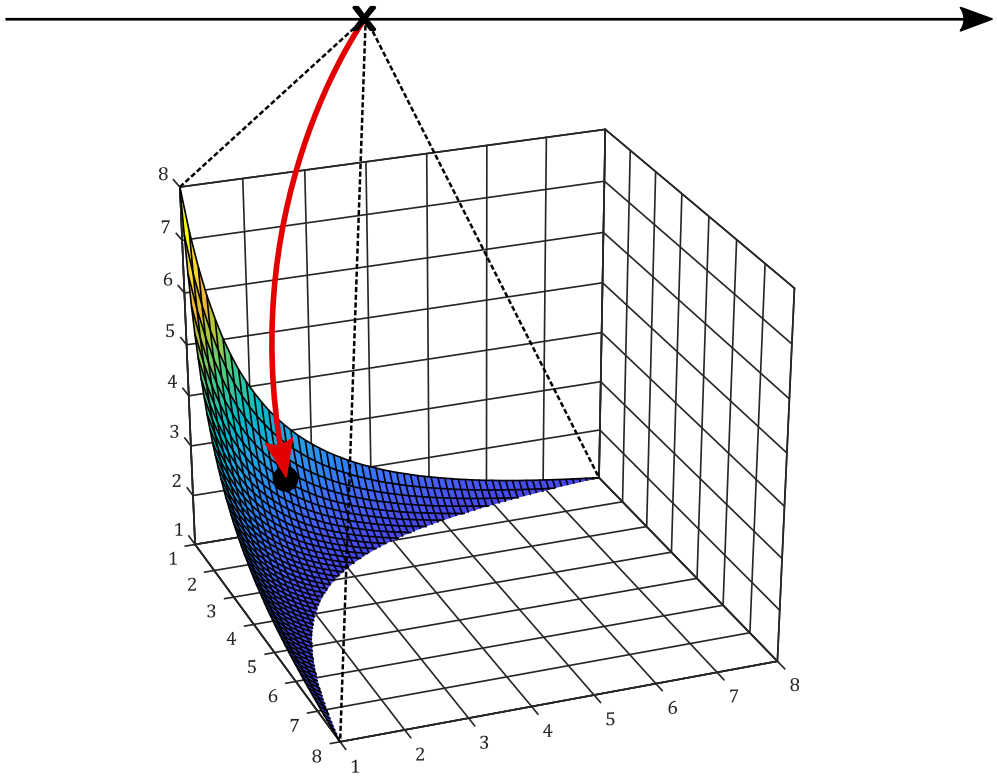
TU Clausthal




Stefan Wittek

# Konsistente, Verteilungskonforme Multi-Level-Simulation auf Basis gelernter, nicht-determinierter Abstraktionsübergänge

ISSE | Institute for Software and Systems Engineering

# Konsistente, Verteilungskonforme Multi-Level-Simulation auf Basis gelernter, nicht-determinierter Abstraktionsübergänge

D i s s e r t a t i o n

zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt von
M.Sc. Stefan Wittek
aus Oelde

genehmigt von der Fakultät für
Mathematik/Informatik und Maschinenbau
der Technische Universität Clausthal

Tag der mündlichen Prüfung
03.05.2021

Dissertation der Technischen Universität Clausthal

2021

**Dekan**

Prof. Dr.-Ing. Volker Wesling

**Vorsitzender der Promotionskommission**

Prof. Dr. Olaf Ippisch

**Betreuer**

Prof. Dr. Andreas Rausch

**Gutachter**

Prof. Dr. Jens Grabowski

D 104

# Kurzzusammenfassung

Die Entwicklung komplexer Systeme, wie Fabrikanlagen, autonome Schienenfahrzeuge oder Flugzeuge, wird fast ausnahmslos von Modellierung und Simulation begleitet. Die Methode erlaubt es in allen Phasen der Entwicklung, Entwürfe zu evaluieren und Alternativen gegenüber zu stellen.

Für jedes Modell muss entschieden werden, wie detailliert es das System abbildet und wie groß sein Betrachtungsausschnitt ist. Typischerweise unterliegt diese Entscheidung einem beschränkten Projektbudget, sodass die Dimensionen Detailgrad und Ganzheitlichkeit in der Praxis gegeneinander abgewogen werden müssen. Ein Modell des gesamten Systems kann somit nur in einem bestimmten, maximalen Detailgrad realisiert werden (Grobebene). Um eine detaillierte Betrachtung zu ermöglichen, muss ein kleinerer Betrachtungsgegenstand gewählt werden (Detailebene). Somit reduziert sich entweder der Detailgrad oder die Perspektive.

Eine in der Literatur viel diskutierte Lösung dieses scheinbaren Dilemmas liegt darin, die Modelle der Grob- und der Detailebene zu verbinden. Es entsteht ein Hybrid, der genau da detailliert ist, wo es für eine bestimmte Fragestellung erforderlich ist. Die Modelle sind jedoch auf unterschiedlichen Abstraktionsebenen angesiedelt. Um sie zu verbinden, müssen Übergange (z.B. Adapter) zwischen ihnen geschaffen werden. Bei dem Übergang von der Detailebene in die Grobebene ($up$) geht typischerweise Information verloren, die im umgekehrten Fall ($down$) wieder „aufgefüllt“ werden muss. Dieses „Auffüllen“ ist im Allgemeinen nicht eindeutig. Dennoch fordern bestehende Ansätze häufig, dass die Verkettung von $down$ und $up$ (schwache Konsistenz) sowie von $up$ und $down$ (starke Konsistenz) die Identität liefert. Hierdurch wird $down$ eindeutig gemacht. Im Rahmen dieser Arbeit wird anhand eines einfachen Lieferkettenbeispiels aufgezeigt, dass diese Eindeutigkeit zu großen Problemen für die Validität der Simulation führen kann.

Um dieses Problem aufzulösen, wird in der vorliegenden Arbeit die gekoppelte Simulation beider Ebenen als indeterministisch verstanden. Es wird gefordert, dass sie der (schwachen) Konsistenz genügt und einer geeignete Wahrscheinlichkeitsverteilung folgt.

Ein weiteres Problem bestehender Ansätze liegt darin, dass sie den Effizienzgewinn aus der Verbindung beider Ebenen durch zusätzlichen Modellierungsaufwand für $up$ und $down$ gefährden.

Das Ziel der vorliegenden Arbeit ist somit der Entwurf einer Architektur für lernende Multi-Level-Simulationen. Lernend deshalb, da hierbei konsistente und verteilungskonforme Übergänge mithilfe von maschinellen Lernverfahren basierend auf Beispielen generiert werden.

# Danksagungen

Eine Dissertation ist eine große und zeitintensive Aufgabe, die man zwar eigenständig jedoch glücklicherweise nicht allein bewältigen muss. Aus diesem Grund möchte ich an dieser Stelle alle jenen Menschen danken, welche mich bei dieser Aufgabe unterstützt haben.

Ich danke meinem Betreuer Prof. Dr. Andreas Rausch, für die vielen Stunden fachlicher Diskussion, Anleitung und Kritik, aber auch für die Gelegenheit als wissenschaftlicher Mitarbeiter an seinem Lehrstuhl tätig zu sein.

Ich danke allen meinen Kollegen am Institut für Software and Systems Engineering, welche mit mir über die Jahre zusammengearbeitet haben, für das offene Klima und die vielen anregenden und konstruktiven Gespräche.

Ich bedanke mich bei dem Simulationswissenschaftlichen Zentrum Clausthal-Göttingen für die Förderung des Projektes „Multi-Level-Simulation“ in welchem ich in großen Teilen meiner Promotionszeit beschäftig war, sowie für die Gelegenheiten zur wissenschaftlichen Diskussion im Rahmen von Veranstaltungen dieses Zentrums.

Ich danke Prof. Dr. Jens Grabowski für seinen Beitrag als Gutachter dieser Dissertation. Zudem danke ich ihm sowie Johannes Erbel und Michael Göttsche, für die intensive Zusammenarbeit in unserem Projekt und die daraus erwachsenen Impulse für meine Promotion.

Ich danke meiner Frau Sabrina, meinen Kindern Vincent und Melissa sowie meiner Familie für all die Zeit, welche ich in dieses Herzenzprojekt stecken durfte.

Schließlich danke ich meinem Vater, der leider viel zu früh gestorben ist. Ja, es macht immer noch Spaß!

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

Komplexe Systeme, wie Netzwerke automatisierter Fabrikanlagen, autonome Schienenfahrzeuge oder Flugzeuge, setzen sich aus einer Vielzahl von Komponenten und einer Hierarchie von Subsystemen zusammen. In der Literatur finden sich die Begriffe System of Systems (Maier, 1998), Cyber Physical Systems (Lee, 2008) oder vernetzte mechatronische Systeme (VDI, 2003) für diese komplexen Systeme. Allen gemeinsam ist die teilweise enge Interaktion zwischen den diversen Komponenten, welche gemeinsam ein gewünschtes Verhalten erfüllen. Autonome Schienenfahrzeuge und Flugzeuge bringen beispielsweise ihre Passagiere sicher an ihr Ziel. Fabrikanlagen stellen effizient Produkte her.

Die Entwicklung dieser komplexen Systeme erfordert es, sowohl deren Komponenten als auch die Interaktion dieser Komponenten zu gestalten. Hierbei bietet Simulation eine Vielzahl von Potenzialen. Der Entwicklungsprozess wird im Idealfall kontinuierlich durch Simulation begleitet (z.B. (VDI, 2003)). In frühen Phasen können grobe Entwurfsalternativen gegenübergestellt, später detaillierte Wirkprinzipien einzelner Komponenten untersucht und deren dynamische Eigenschaften analysiert werden. In virtuellen Prototypen (Wang, 2002) und mit Hardware-in-the-loop-Studien (Isermann, 1999) wird schließlich die Interaktion der Komponenten und das Verhalten des integrierten Systems mithilfe von Simulation analysiert. Da so der Aufwand für den Aufbau physikalischer Prototypen reduziert wird, können Kosten gespart werden.

Zudem kann Simulation selbst zu einem Teil der Systeme werden. Virtuelle Sensoren und modellprädiktive Regler erlauben etwa die Erfassung von System- und Umgebungszuständen, ohne entsprechende Sensoren in das System zu integrieren (Camacho, 2007). Durch dieses indirekte Messen können entweder nicht direkt messbare Größen erfasst oder Kosten durch eine Reduktion der Sensorausstattung eingespart werden.

Um diese Potentiale heben zu können, müssen die simulierten Modelle ganzheitlich genug sein, um alle relevanten Komponenten des Systems zu beinhalten, dürfen dabei allerdings nur so komplex sein, wie es Modellierung- und Rechenressourcen zulassen. Dies zwingt Modellierer dazu, zwischen dem Detailgrad des Modells und der Weite des betrachteten Systemausschnittes (Ganzheitlichkeit) abzuwägen.

> *"Given a fixed amount of resources (time, space, personnel, etc.), and a model complexity that exceeds this limit, there is a trade-off relation between size and resolution. We may be able to represent some aspects of a system very accurately but then only a few components will be representable. Or we may be able to provide a comprehensive view of the system but only to a relatively low resolution."* (Zeigler, 2019, S. 409)

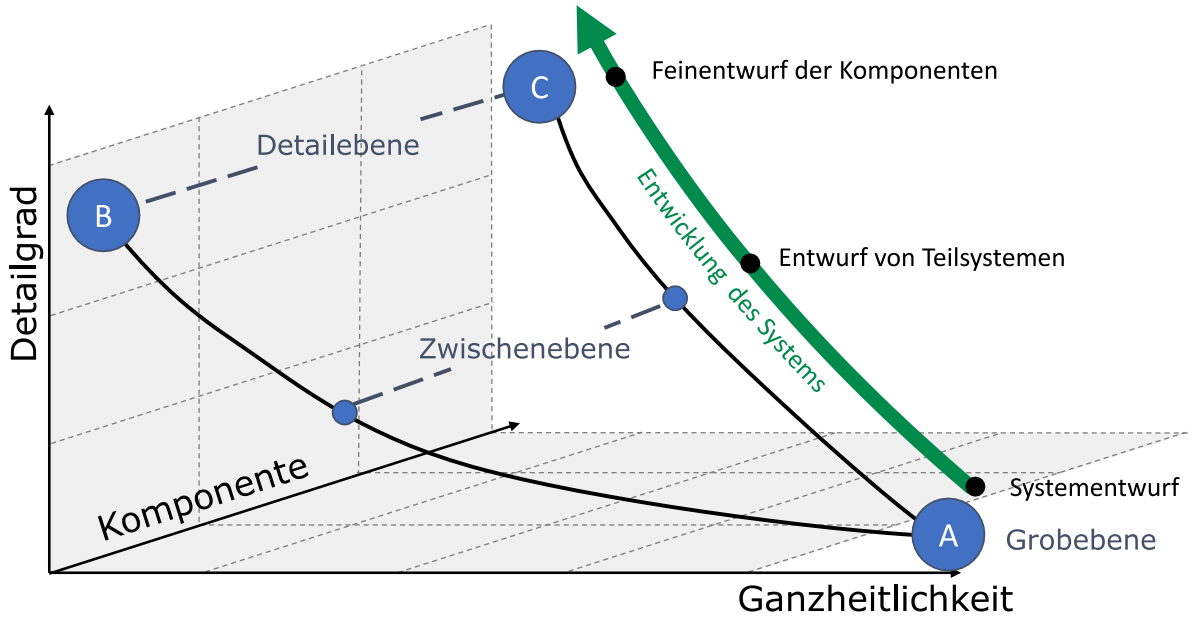



*Abbildung 1.1: Abwägung zwischen Detailgrad und Ganzheitlichkeit* (Wittek, 2018).

Dieses Modell findet sich in der Abbildung als Punkt A wieder. Um detailliertere Untersuchungen anzustellen, ist der Modellierer gezwungen, den betrachteten Ausschnitt zu reduzieren. Er gelangt über eine Zwischenebene, mit einzelnen Teilsystemen, zu einer Detailebene mit dem Modell B, welches nur eine einzelne Komponente enthält. Dafür ist es jedoch möglich, diese mit großem Detailgrad abzubilden (Detailebene). Natürlich kann der Modellierer auch jede andere Komponente auf diese Weise detailliert modellieren. Dies ist jedoch immer nur isoliert möglich, da bereits eine dieser detaillierten Komponenten die vorhandenen Ressourcen ausschöpft. Die Wahl der Komponenten bildet schließlich die dritte Dimension der Abbildung. Sie wird als z-Achse dargestellt.

Typsicherweise folgt der Entwicklungsprozess komplexer Systeme einem ähnlichen Zerlegungskonzept. Zunächst wird ein grober Entwurf des gesamten Systems erstellt, welcher die Komponenten des Systems und deren Schnittstellen definiert. Später werden diese Komponenten ausgestaltet. Abschließend wird das System aus diesen Komponenten integriert (vgl. (VDI, 2003)).

Durch dieses Zoomen auf einzelne Komponenten ist es zwar möglich, detailliert die Eigenschaften der einzelnen Komponenten zu analysieren, durch die isolierte Betrachtungsweise geht jedoch die Interaktion zwischen den Komponenten und der ganzheitliche Blickwinkel verloren. Dieser ist jedoch im Sinne einer virtuellen Integration notwendig. Es ist nicht garantiert, dass sich die lokal optimierten Komponenten zu einem

global-optimalen System zusammenfügen. Dies kann im schlechtesten Fall erst bei der physikalischen Integration durch den Aufbau eines Prototyps erkannt werden.

Eine viel diskutierte Möglichkeit, die ganzheitliche Perspektiv der Grobebene mit dem hohen Detailgrad der Detailebene zu kombinieren, liegt darin, die Modelle unterschiedlicher Ebenen zu verbinden. Die vorliegende Arbeit untersucht hierzu das Konzept der Multi-Level-Simulation. Hierbei werden die Modelle der Ebenen direkt miteinander gekoppelt. Es entsteht ein Hybrid, der genau da detailliert ist, wo es für eine bestimmte Fragestellung erforderlich ist. Auf diese Weise werden die Ressourcen effizient eingesetzt.

Das System wird bei einer Multi-Level-Simulation auf mehreren Ebenen gleichzeitig simuliert. Da die Modelle auf unterschiedlichen Abstraktionsebenen angesiedelt sind, gilt dies auch für ihre Datentypen und Zustandsräume. Um die Simulationen zu verbinden, müssen Übergänge (z.B. Adapter) zwischen ihnen geschaffen werden. Bei dem Übergang von der Detailebene in die Grobebene ($up$) geht typischerweise Information verloren. Der umgekehrte Übergang ($down$) muss diese wieder „auffüllen“. Diese Übergänge stellen die zentrale Herausforderung einer Multi-Level-Simulation da.

## 1.1 Ziele

In der Literatur finden sich eine Vielzahl von Arbeiten, welche sich mit den Übergängen zwischen Simulationen verschiedener Abstraktionsebenen befassen. Viele dieser Autoren legen besonderen Fokus auf Konsistenzanforderungen zwischen den Übergängen. Es wird gefordert, dass die Verkettung von $down$ und $up$ (schwache Konsistenz) sowie von $up$ und $down$ (starke Konsistenz) die Identität liefert.

Im Laufe dieser Arbeit wird anhand eines anschaulichen Beispiels verdeutlicht, dass die starke Konsistenz zu verzerrten Simulationsläufen führt. Sie impliziert, dass $down$ die Umkehrfunktion von $up$ ist. Somit ist $down$ insbesondere eindeutig. Wir werden sehen, dass hierin eine zentrale Limitation der Ansätze liegt.

Um diese Limitation zu überwinden, wird $down$, also das „Auffüllen“ der Information, in der vorliegenden Arbeit als nicht-determiniert verstanden. Zudem fordern wir, dass dieses „Auffüllen“ der Informationen einer geeigneten Wahrscheinlichkeitsverteilung folgt. An die Stelle der starken Konsistenzanforderung tritt eine Verteilungsanforderung.

Keiner der bestehenden Ansätze ist in der Lage, die (schwache) Konsistenzanforderung und die Verteilungsanforderung zu erfüllen (siehe Kapitel 5).

Zudem verlangen fast alle dieser Ansätze, dass der Modellierer die Übergänge manuell definiert. Hierdurch entsteht jedoch zusätzlicher Modellierungsaufwand, der den Effizienzgewinn einer Multi-Level-Simulation hinsichtlich des Modellierungsaufwands bedroht.

Aus diesem Grund ist das Ziel dieser Arbeit der Entwurf einer Architektur für lernende Multi-Level-Simulation. Lernend deshalb, da hierbei konsistente und verteilungskonforme Übergänge mithilfe von maschinellen Lernverfahren basierend auf Beispielen automatisch generiert werden.

Um dieses Ziel zu erreichen, wird diese Arbeit die folgenden Beiträge erbringen:

- Eine Architektur für Multi-Level-Simulation (MLS) wird definiert und formalisiert.
- Das Zusammenspiel von Abstraktion mit Konsistenz und Verteilung bei den Übergängen einer MLS wird formalisiert und in zwei zentrale Anforderungen (Konsistenz und Verteilung) zusammengefasst.
- Für die Generierung der Übergänge werden bekannte Verfahren des maschinellen Lernens zu einem neuartigen hybriden Verfahren kombiniert. Dieses erreicht sowohl Konsistenz als auch eine korrekte Verteilung.
- Es werden zwei Fallstudien zur Evaluation der Ergebnisse durchgeführt.

## 1.2 Kapitelübersicht

Im Folgenden werden die Kapitel der Arbeit sowie deren Inhalt kurz vorgestellt.

In Kapitel 2 werden zunächst die Grundlagen der Arbeit behandelt. Hierbei werden die Begriffe Modell und Simulation beleuchtet sowie technische Lösungen zu deren Kopplung vorgestellt. Zudem wird eine kurze Einführung in die Wahrscheinlichkeitstheorie - wie sie im weiteren Verlauf für die Definition der Verteilungsanforderung notwendig ist - gegeben. Abschließend wird exemplarisch an mechatronischen Systemen die Integration von Simulation in den Entwicklungsprozess vorgestellt.

In Kapitel 3 wird die Problematik einer Kopplung von Modellen unterschiedlicher Ebenen anhand des anschaulichen Beispiels einer Lieferkette beleuchtet. Dabei wird die Planung dieser Lieferkette mit verschiedenen Standorten besprochen. Anhand des Beispiels werden möglich Lösungsansätze für eine Verbindung von zwei Abstraktionsebenen beschrieben sowie die hierbei auftretenden prinzipiellen Probleme diskutiert. Hierbei wird abschließend auch eine naive Kopplung der Modelle mithilfe von Adaptern beschrieben, wobei die Limitation eines determinierten $down$ deutlich wird.

In Kapitel 4 wird diese Limitation genauer analysiert. Hierzu wird zunächst vor dem Hintergrund des Beispiels das Auftreten relevanter Abstraktionsbeziehungen aufgezeigt. Anschließend werden die Konsistenz- und die Verteilungsanforderung an eine Multi-Level-Simulation formuliert.

In Kapitel 5 werden bestehende Ansätze zur Simulation in verschiedenen Abstraktionsebenen vorgestellt und in Hinblick auf die beiden Anforderungen bewertet. Bei fast allen dieser Ansätze liegt jedoch ein starker Fokus auf der Konsistenzanforderung. Die Verteilungsanforderung wird meist nicht betrachtet und stattdessen von einer Dirac-Verteilung (Eindeutigkeit) ausgegangen.

Ausgehend hiervon können die Forschungsfragen, welche im Rahmen dieser Arbeit untersucht werden, aufgestellt werden. Dies geschieht in Kapitel 6.

Um die nun offensichtliche Lücke zu schließen, wird in Kapitel 7 eine formale Architektur für Multi-Level-Simulation vorgestellt. Auch die beiden Anforderungen an die Übergänge werden hierbei betrachtet und auf Anforderungen an $up$ und $down$ heruntergebrochen.

Kapitel 8 liefert den Rahmen für das automatische Generieren der Übergänge $up$ und $down$ mithilfe von maschinellen Lernverfahren. Zudem wird ein Konzept zur Bewertung der Genauigkeit, bzw. des Fehlers der Übergänge erarbeitet.

In Kapitel 9 wird nach einem konkreten Verfahren für die Bestimmung von $up$ gesucht. Das Ergebnis wird anhand einer Reihe von Beispielen aus der Literatur evaluiert.

Anschließend wird in Kapitel 10 ein maschinelles Lernverfahren zur Bestimmung von $down$ ermittelt. Das zugrundeliegende Lernproblem entspricht dem Schätzen einer bedingten Wahrscheinlichkeitsdichte, von der zudem Stichproben generiert werden können. Um hierbei sowohl Konsistenz als auch Verteilung zu erreichen, wurden zwei Verfahren zu einem neuen, hybriden Verfahren kombiniert. Auch dieses Verfahren wird in diesem Kapitel evaluiert.

Kapitel 11 beschreibt die Implementierung einer Multi-Level-Simulationsplattform. Diese setzt die formale Architektur um. Durch eine Integration der Simulationsumgebung Ptolemy können Modelle angeschlossen werden. Schließlich wird dargestellt, wie $up$ und $down$ integriert werden.

In Kapitel 12 wird die Evaluation der Ergebnisse in zwei Fallstudien vorgestellt. Hierzu werden diese Fallstudien mit der Multi-Level-Simulationsplattform umgesetzt, die Übergänge aus Beispielen gelernt und anschließend die Simulationsläufe untersucht.

Kapitel 13 liefert eine abschließende Zusammenfassung der Arbeit sowie einen Ausblick auf mögliche zukünftige Arbeiten.

# 2 Grundlagen

## 2.1 Modelle und Simulation

Der Begriff Simulation ist, genau wie die eng damit verbundenen Begriffe Modell und Simulator, von zentraler Bedeutung für die im Rahmen dieser Arbeit untersuchten Fragestellungen. Dieser Abschnitt stellt die Begriffe anhand des Rahmenwerks für Modellierung und Simulation nach Zeigler (Zeigler, 2019, S. 27ff) gegenüber.

Hierbei steht zunächst der Simulationsbegriff im Zentrum der Betrachtungen. Der Verein Deutscher Ingenieure (VDI) definiert **Simulation** wie folgt:

> *„Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind."*(VDI, 2008, S. 2)

In dieser Definition steht hinter der Simulation der Zweck Experimente durchzuführen. Dieser Zweck spielt auch bei Ören eine zentrale Rolle (Ören, 2011).

> *„Simulation is performing goal directed experiments with models of dynamic systems."* (Ören, 2011, S. 44)

Diese Experimente können wiederum zur Entscheidungsunterstützung, zum allgemeinen Erkenntnisgewinn oder für die Bildung durchgeführt werden (Ören, 2011). Insbesondere die Entscheidungsunterstützung ist dabei für diese Arbeit von besonderem Interesse. Für Ören fallen unter diesen Zweck Aktivitäten wie beispielsweise die Vorhersage des Verhaltens und die Bewertung von Systementwürfen sowie das Auswerten virtueller Prototypen.

Von dem Zweck Experimente durchzuführen, kann der Zweck Erfahrungen zu vermitteln unterschieden werden.

> *„Simulation is providing experience under controlled conditions"* (Ören, 2011, S. 45)

Diese Erfahrungen ermöglichen das Training von Fähigkeiten in einer sicheren Umgebung. So müssen etwa Piloten ihre Fähigkeit, ein Flugzeug zu steuern, zunächst an einem virtuellen Simulator unter Beweis stellen, bevor sie zu Lehrflügen den Boden verlassen dürfen (Ören, 2011).

Ein weiterer Zweck, der sich aus dem Vermitteln von Erfahrungen ableitet, besteht in der Unterhaltung. Ein Beispiel hierfür sind simulationsbasierte Computerspiele, um die sich eine ganze Industrie aufgebaut hat (Zackariasson, 2012).

In allen genannten Perspektiven wird Simulation dann genutzt, wenn Experimente und Erfahrungen an den realen Objekten nicht möglich oder impraktikabel sind, etwa aufgrund von Kosten sowie damit verbundenen Gefahren.

Die Anwendung von Simulation innerhalb des Entwicklungsprozesses liefert den Rahmen für diese Arbeit. Entsprechend wird hier Simulation vorwiegend, aus der Perspektive der Durchführung von Experimenten betrachtet.

Diese Perspektive liegt auch der „Theory of Modeling and Simulation“ von Bernard P. Zeigler (Zeigler, 2019) zugrunde. Abbildung 2.1 zeigt das in dieser Arbeit beschriebene Rahmenwerk für Modellierung und Simulation.

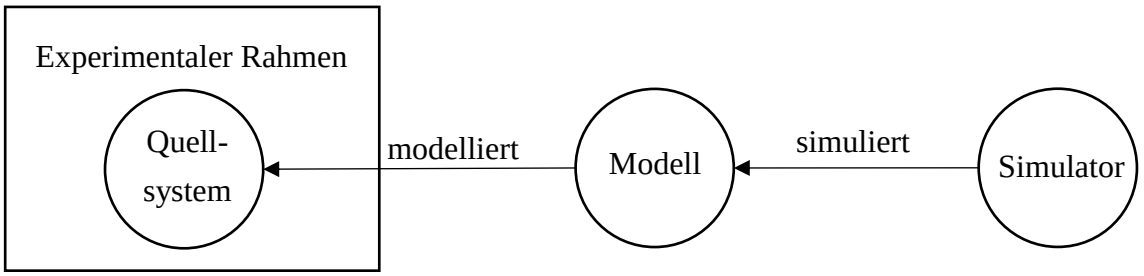



*Abbildung 2.1: Modellierung und Simulation nach Zeigler* (Zeigler, 2019, S. 28).

Das **Quellsystem** steht für den Betrachtungsgegenstand einer Simulation. Das dynamische Verhalten dieses Quellsystems wird beobachtet und gemessen. Um diese Beobachtungen durchführen zu können, werden Experimente mit dem Quellsystem durchgeführt. In diesen Experimenten wird das System auf eine definierte Art und Weise stimuliert und die Messtrajektorien bestimmter Eigenschaften aufgezeichnet.

Wie genau diese Experimente durchgeführt werden und welche Eigenschaften gemessen werden, wird in einem **experimentellen Rahmen** spezifiziert. Der experimentelle Rahmen operationalisiert das Ziel einer Simulation. Entsprechend kann ein Quellsystem mit unterschiedlichen experimentellen Rahmen untersucht werden.

So könnte das Ziel einer Simulation darin bestehen, die Durchlaufzeit einer Sendung durch ein Hochregallager zu bestimmen. In diesem Fall würde ein experimenteller Rahmen eine repräsentative Abfolge von Bestellungen und Wareneingängen enthalten. Bei der Beobachtung der Warenabgänge müsste die Zuordnung zu Bestellungen miterfasst werden. Ist das Ziel der Simulation hingegen, festzustellen, wie schnell das Lager geleert werden kann, legt der experimentelle Rahmen fest, dass alle Waren im Lager gleichzeitig abgefragt werden und dass in diesem Fall nur der Füllstand des Lagers gemessen werden muss.

Das **Modell** definiert Ziegler wie folgt:

> *„...[A model is] a set of instructions, rules, equations, or constraints for generating I/O behavior."* (Zeigler, 2019, S. 32)

In dieser Betrachtung wird das Modell zu einer Systemspezifikation. Legt man einen Startzustand des Modells und eine Abfolge von Eingaben fest, kann mithilfe der im Modell zusammengefassten Anweisungen, Regeln, Gleichungen oder Bedingungen eine Abfolge von Systemzuständen und Ausgaben berechnet werden.

Schließlich gibt es, getrennt vom Modell, den **Simulator**. Dieser agiert als eine Art Ausführungsumgebung für die Anweisungen des Modells. Zeigler definiert den Simulator wie folgt:

> *"... a simulator is any computation system [...] capable of executing a model to generate its behavior."* (Zeigler, 2019, S. 32)

Um eine sinnvolle Simulation durchführen zu können, müssen zwischen diesem Quellsystem, Modell und Simulator die folgenden Beziehungen gelten.

Ein Modell **modelliert** ein Quellsystem **valide**, wenn es mit einer festgelegten Fehlertoleranz in der Lage ist, das Verhalten des Quellsystems in den spezifizierten Experimenten nachzuahmen. Man spricht in diesem Zusammenhang auch von Validität des Modells. Diese Beziehung zwischen dem Quellsystem und dem Modell ist dabei nicht als zeitliche oder kausale Abfolge zu verstehen. Das Quellsystem ist also nicht die Quelle des Modells, sondern die Quelle von Messdaten, mit denen die Validität des Modells nachgewiesen werden kann. Ein Modell kann auch ein noch nicht-existierendes, geplantes Quellsystem modellieren. Dies ist beispielsweise im Rahmen des Produktentwicklungsprozesses notwendig (VDI, 2003). In diesem Fall kann die Validität des Modells erst abschließend und unter Verwendung von Prototypen als Quelle für Messdaten nachgewiesen werden.

Dennoch können auch bei der Entwicklung z.B. valide Teilmodelle gekoppelt werden, um früh mit einer relativen Sicherheit Alternativen bewerten zu können. Wie Simulation in Produktentwicklungsprozessen eingesetzt werden kann, wird in Abschnitt 2.4 näher beleuchtet.

Um ein Modell zu **simulieren**, muss der Simulator in der Lage sein, aus einer Eingangssequenz und einem gegebenen Startzustand die entsprechenden Ausgaben des Modells zu generieren. Gelingt dies garantiert, spricht man davon, dass zwischen dem Simulator und dem Modell eine **korrekte Simulationsbeziehung** besteht.

### 2.1.1 Arten von Modellen

Im Laufe der Zeit und in verschiedenen Wissensgebieten wurden eine Vielzahl von Modellarten entwickelt. In diesem Abschnitt werden grundlegende Modellarten vorgestellt.

Eine zentrale Eigenschaft von Modellen ist ihr Zeitmodell. Es legt fest, wie die Zeit in den Modellen vergeht. Das Zeitmodell diktiert, wie die Beschreibung des Modells aussehen muss und wie der Simulator das Verhalten des Systems erzeugen kann (indem er Zeit vergehen lässt). Entsprechend ist es verbreitet, die verschiedenen Zeitmodelle zu nutzen, um Modellarten zu definieren. Ziegler unterscheidet beispielsweise drei fundamentale Modellarten, die er jeweils mit einem Zeitmodell verbindet (Zeigler, 2019). Im Folgenden werden diese drei grundlegende Modellarten (zeitdiskret, ereignisdiskrete und differenzialgleichungsbasierte Modelle) vorgestellt. Zudem wird die Kombination aus differenzialgleichungsbasierten und ereignisdiskreten Modellen, die sogenannten hybriden Differenzialgleichungsysteme, vorgestellt.

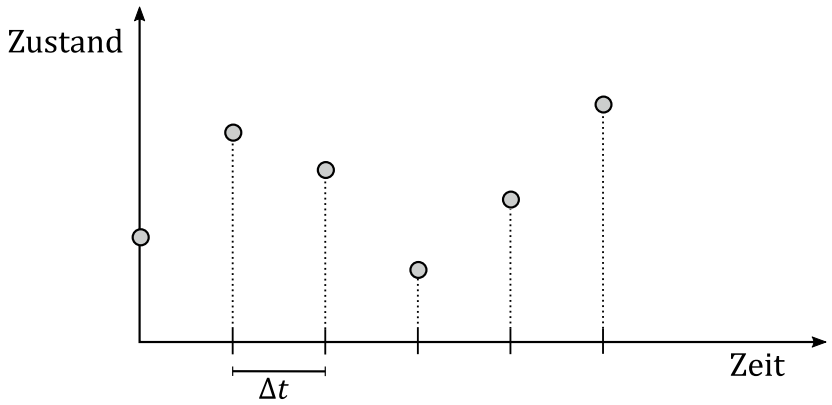



*Abbildung 2.2: Zeitverlauf des Zustandes eines zeitdiskreten Systems (DT).*

Betrachten wir zunächst die sogenannten zeitdiskreten (**Discrete Time – DT**) Modelle. Sie spezifizieren DT-Systeme, die sich dadurch auszeichnen, dass die Zeit sich ausschließlich in äquidistanten Zeitschritten ändert. Die Bedeutung der Länge eines solchen Zeitschrittes $\Delta t$ kann dabei je nach Modell unterschiedlich sein, z.B. eine Sekunde, eine Stunde oder ein Jahr. Bei seiner Ausführung durchläuft das System eine Abfolge gleichweit voneinander entfernter Zeitpunkte und es befindet sich in jedem dieser Zeitpunkte in einem Zustand. Eine Übergangsfunktion ermittelt aus dem aktuellen Zustand und dem Input, den das System zu diesem Zeitpunkt erhält, einen Folgezustand. Typischerweise wird diese Übergangsfunktion explizit angegeben. Abbildung 2.2 zeigt schematisch, wie sich der Zustand eines DT-Modells verändert, während es simuliert wird. Da die Zeitachse den natürlichen Zahlen gleichzusetzen ist, ist der Zustand an diesen Punkten, jedoch nicht dazwischen, definiert. Der Wertebereich des

Zustands kann in DT-Modellen sowohl diskret als auch kontinuierlich sein. Die Abbildung deutet einen kontinuierlichen Wertebereich (etwa $\mathbb{R}$) an.

Ein Beispiel für DT-Modelle sind zellulärer Automaten (Wolfram, 1984) die etwa zur Beschreibung von biologischen Phänomenen (Mallet, 2007) oder als Grundlage für Verkehrssimulation (Nagel, 1992) genutzt werden. Darüber hinaus nutzen Werkzeuge zur Simulationskopplung oft vergleichbare Zeitmodelle für die Teilmodelle (z.B. FMI, siehe Abschnitt 2.2.1).

In ereignisdiskreten Modellen (**Discrete Event - DE**) ändert sich der Zustand des Systems ausschließlich als Reaktion auf Ereignisse (Events). Beispiele für ein solches Ereignis sind die Ankunft einer Nachricht, das Abschließen eines Fertigungsauftrags oder das Versagen einer Netzwerkkomponente. Jedem dieser Ereignisse wird ein Zeitstempel zugeordnet. Diese Zeitstempel können an einer beliebigen Stelle des realwertigen Zeitstrahls liegen. Die Liste aller zukünftig geplanten Ereignisse wird dabei als Teil des Zustandes verstanden.

Der Simulator wählt mithilfe einer *Scheduling* Funktion das nächste Ereignis aus und springt mit der Systemzeit zu dem mit dem Ereignis assoziierten Zeitpunkt. Es ist hierbei auch möglich, dass Ereignisse auf denselben Zeitpunkt fallen. Wie mit dieser Situation umgegangen wird, unterscheidet sich in verschiedenen Varianten der DE-Modelle. So kann etwa eine statische Priorität vergeben werden, oder beispielsweise eine zustandsabhängige Funktion, der sogenannte *tie-breaker*, genutzt werden. Auch eine parallele Verarbeitung ist möglich.

Im Modell sind für alle Ereignisse Verarbeitungsroutinen definiert. Diese erzeugen einen neuen Zustand auf Basis des aktuellen Zustands. Diese Routinen können auch neue Ereignisse erzeugen. Zudem kann es sein, dass bereits geplante Ereignisse nicht mehr gültig sind. Fällt etwa in einem Rechnernetz ein Berechnungsserver als Ergebnis eines Unfall-Events aus, müssen alle geplanten Ereignisse, mit denen dieser Server Zwischenergebnisse kommuniziert hätte, wieder gelöscht werden.

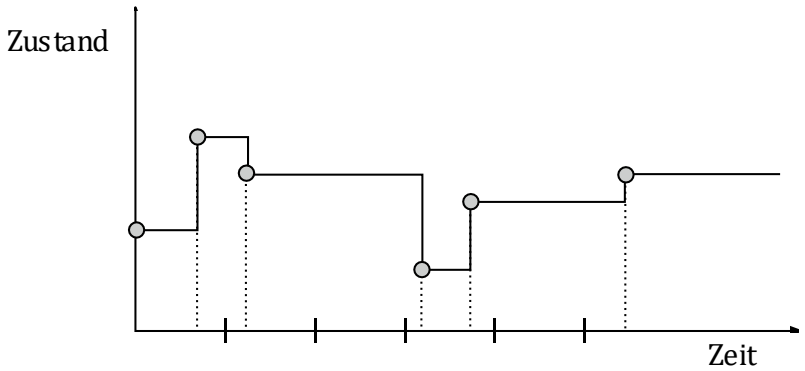



*Abbildung 2.3: Zeitverlauf des Zustandes eines ereignisdiskreten Systems (DE).*

Abbildung 2.3 zeigt wie die Zeit in ereignisdiskreten Systemen voranschreitet. Wie schon bei DT-Systemen, ändert sich der Zustand lediglich an diskreten Zeitpunkten, allerdings sind diese bei DE nicht gleichverteilt. Zudem ist der Systemzustand zwischen zwei Ereignissen definiert als der Zustand nach dem letzten Ereignis. Dies ergibt sich aus der Realwertigkeit der Zeitachse. Der Zustandsraum kann erneut diskret oder kontinuierlich sein. Beispiele für Systeme, die mit DE-Modellen modelliert werden, sind Logistiksysteme (Tako, 2012), betriebswirtschaftliche Systeme (Sato, 1997) oder Systeme aus dem Gesundheitsbereich (Jun, 1999).

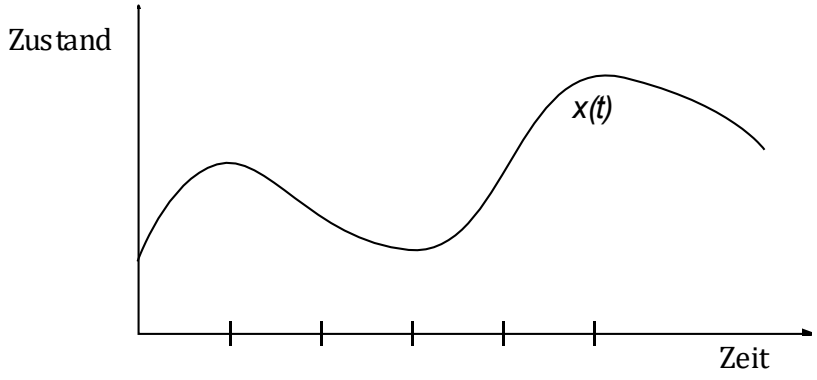



*Abbildung 2.4:Zeitverlauf des Zustands eines DG-Systems.*

Modelle aus **Differenzialgleichungen** (**DG**) ermöglichen es, Systeme mit kontinuierlicher Zeit zu modellieren. Der Zustand des Systems ist dabei eine kontinuierliche Funktion über der Zeit. Für viele physikalische Phänomene ist es leichter – oder sogar der einzige Weg – diese Funktionen nicht direkt zu formulieren, sondern als Funktionen von Änderungsraten – also Differentialen – dieser Zustandsfunktion indirekt zu beschreiben (Cellier, 1991).

Ein einfaches Beispiel dafür, wie ein System mithilfe von Differenzialgleichungen beschrieben werden kann, ist ein Auto, das sich entlang einer geraden Straße bewegt. Die Position des Autos über der Zeit x(t) in $m$ ist der Zustand von Interesse. Die Geschwindigkeit des Fahrzeugs in $\frac{m}{s}$ ist genau die Änderungsrate der Position, also die Ableitung der Position nach der Zeit $\frac{dx(t)}{dt}$. Da aus dem Kontext klar ist, dass Ableitungen nach der Zeit betrachtet werden, wird hier oft die Schreibweise $\dot{x}(t) = \frac{dx(t)}{dt}$ genutzt. Nehmen wir an, in unserem Beispiel bewegt sich das Fahrzeug mit konstanter Geschwindigkeit $v = 5\frac{m}{s}$. Kennen wir nun die Startposition des Fahrzeugs, $x(0) = 0$, können wir unser kleines System wie folgt als Differenzialgleichungssystem beschreiben:

$$\dot{x}(t) = v$$

$$x(0) = 0$$

Da wir die Ableitung und einen Anfangswert kennen, spricht man hier von einem Anfangswertproblem. Um nun die Position für einen bestimmten Zeitpunkt t=10 zu ermitteln, muss lediglich das Integral von $\dot{x}(t)$ bestimmt werden. Für unser einfaches Beispiel ist dies analytisch leicht möglich:

$$\int_0^t k = v * t + c = x(t)$$

Mit dem Startwert $x(0) = 0$, können wir die Integrationskonstante c auflösen:

$$x(0) = \ k * 0 + c = 0 \Rightarrow x(t) = k * t$$

$$x(10) = v * 10 = 5\frac{m}{s} * 10s = 50m$$

Nun erhalten wir die explizite Funktion für den Zustand des Systems.

Diese analytische Integration ist in vielen komplexeren Anwendungsproblemen nicht möglich, da hier nicht lineare Differenzialgleichungssysteme entstehen. Nehmen wir also an, wir können auch für dieses Beispiel $\int_0^t k$ nicht analytisch bestimmen.

Es ist auch möglich, ausgehend von der Startposition für jede Sekunde, die vergeht, die Position fortzuschreiben.

$$x(0) = 0$$
$$x(1) = x(0) + 5 = 5$$
$$x(2) = x(1) + 5 = 10$$
$$\ldots$$
$$x(t+1) = x(t) + v$$

Wie bereits erwähnt ist die Zeitachse dieser Modelle kontinuierlich. Wir könnten das Modell also auch in beliebigen Schritten *h* fortschreiben.

$$x(t+h) = x(t) + v * h$$

Nehmen wir nun an, die Geschwindigkeit des Fahrzeuges, $\dot{x}(t)$ ist nicht konstant, sondern ändert sich mit der Zeit, etwa weil der Fahrer beschleunigt oder aufgrund des Luftwiderstandes. So gelangen wir zur folgenden Gleichung:

$$x(t+h) = x(t) + \dot{x}(t) * h$$

Anschaulich bestimmen wir für einen Zeitpunkt $t$, für den wir den Zustand $x(t)$ kennen, die Steigung einer Tangente mit $\dot{x}(t)$. Die Funktion $\dot{x}(t)$ ist, wie eingangs erwähnt, bekannt, da

wir sie explizit modelliert haben. Anschließend gehen wir $h$ Schritte weit entlang dieser Tangente und wählen diesen Punkt als neuen Zustand. Abbildung 2.5 zeigt diese Anschauung.

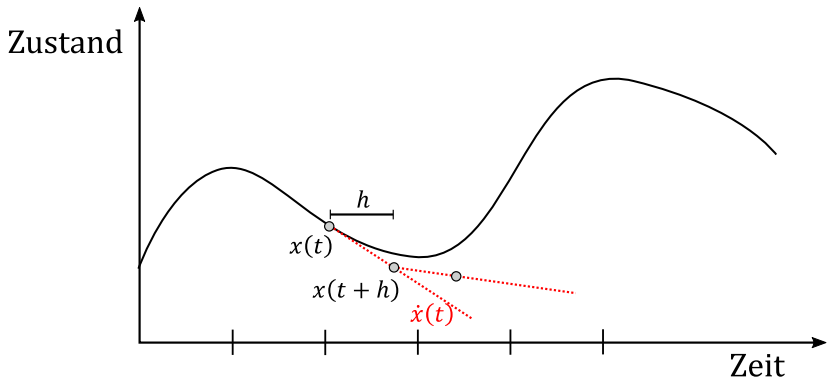



*Abbildung 2.5: Numerische Integration mit Euler Verfahren.*

Das Vorgehen, auf diese Weise einen neuen Punkt der Funktion $x(t+h)$ auf Basis eines Punktes $x(t)$ und dem Differenzial $\dot{x}(t)$ zu berechnen, wird als numerische Integration bezeichnet. Das vorgestellte, einfache Verfahren entspricht dem expliziten Euler-Verfahren. Der Fehler zwischen dieser numerischen Lösung und der tatsächlichen Funktion $x(t)$ kann durch Wahl einer geringeren Schrittweite beliebig klein gehalten werden. Dadurch steigt aber auch der Rechenbedarf, da entsprechend mehr Schritte zum Erreichen eines Zeitpunktes notwendig sind. Ein Kompromiss aus geringem Fehler und Rechenzeit besteht darin, die Schrittweite $h$ dynamisch anzupassen. Zudem ist dieses Euler-Verfahren nur das einfachste Verfahren zur numerischen Integration.

Der Simulator eines DG-Modells führt genau diese numerische Integration aus, um schrittweise die Werte der Funktion $x(t)$ zu bestimmen. Tatsächlich wird numerische Integration und Simulation im Kontext von DG-Modellen von vielen Autoren synonym verwendet.

Ein Beispiel für die Anwendung von DG-Modellen ist die Finite Elemente Methode (Zienkiewicz, 2005), welche zur geometrieabhängigen Untersuchung dynamischer Phänomene wie Kräfteverteilungen, Verformungen oder Wärmeleitung genutzt werden kann.

**Hybride Differenzialgleichungssysteme** (**HDG**) stellen eine Kombination aus DG- und DE-Modellen dar. Diese Kombination ist weit verbreitet. Ziegler beschreibt sie als DEV&DESS (Zeigler, 2019, Kap. 9.3). Auch diese erlaubt es, Systeme mit kontinuierlichem Zustand über einer kontinuierlichen Zeitachse zu beschreiben. Allerdings ist es hier auch erlaubt, dass Ereignisse zu Sprüngen in der Zustandsfunktion führen. Somit ist die Zustandsfunktion nicht mehr kontinuierlich, sondern nur noch stückweise kontinuierlich.

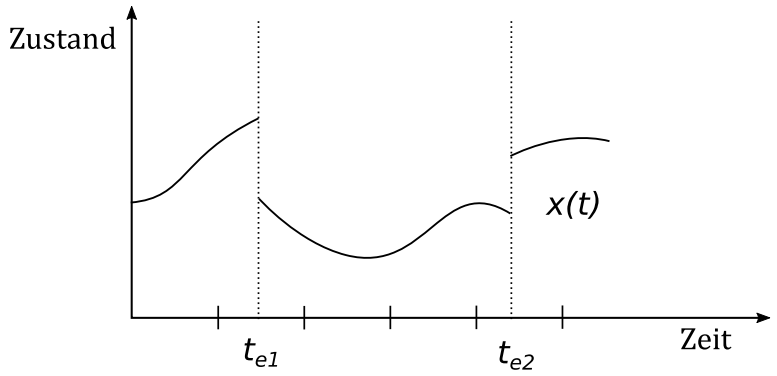



*Abbildung 2.6: Zeitverlauf des Zustands eines HDG-Systems.*

Abbildung 2.6 zeigt den Verlauf des Zustands eines solchen HDG Systems. An den Zeitpunkten $t_{e1}$ und $t_{e2}$ tritt jeweils ein Ereignis auf (e1 bzw. e2). Diese Ereignisse werden in diesen Systemen häufig als Vorzeichenwechsel einer Indikatorfunktion modelliert.

Der Simulator kann die kontinuierlichen Stücke des Zeitverlaufes genau wie bei DG-Modellen über Integrationsschritte berechnen. Hierbei ändert er den Zustand des Systems in diskreten Simulatorschritten variabler Länge, genau wie ein DG-Simulator. Da die Ereignisse einen Sprung der Zustandsfunktion auslösen können, ist es notwendig, dass der Simulator die Zeitpunkte $t_{e1}$ und $t_{e2}$ genau trifft. Hierzu muss er Nullstellen der Indikatorfunktionen finden. Da diese vom zeitkontinuierlichen Zustand des Systems abhängen, ist dies erneut oft nicht analytisch möglich, weshalb der Simulator diese Ereigniszeitpunkte suchen muss. Überschreitet der Simulator mit einem Schritt ein Ereignis, so kann er dies mithilfe der Indikatorfunktion detektieren. Nun muss er den Schritt rückgängig machen, z.B. indem er ein Backup des Zustandes lädt. Anschließend verringert der Simulator die Schrittweite und versucht es erneut. Dies wiederholt er, bis das Ereignis mit einer hinreichenden Genauigkeit gefunden wurde. Diese Genauigkeit muss entsprechend angegeben werden. Werkzeuge wie Matlab Simulink (Simulink, o. J.) oder Modelica (Modelica, o. J.) basieren auf dieser Art von Modell.

## 2.2 Kopplung von Modellen

Wie bereits erwähnt, haben sich in den verschiedenen Wissensgebieten unterschiedliche Arten von Modellen herausgebildet. Neben den vorgestellten Modelltypen existiert eine Vielzahl von Untertypen. Neben dieser Heterogenität auf der semantischen Ebene, kann es für ein und denselben Modelltyp eine Vielzahl von Notationen geben. Beispielsweise sind für DG-Modelle sowohl grafische, als auch textuelle und Mischformen aus beiden Notationen verbreitet. Schließlich ist es auch möglich, dass sich selbst bei gleicher Notation in unterschiedlichen Disziplinen unterschiedliche Softwarewerkzeuge durchgesetzt haben. Um einen größeren Teil des untersuchten Systems beschreiben zu können, werden Modelle miteinander gekoppelt und

simuliert. Hierzu muss diese Heterogenität überwunden werden. Im folgenden Abschnitt werden zwei für diese Arbeit relevante Beispiele hierfür vorgestellt. Dabei setzen die Ansätze auf unterschiedlichen Ebenen an.

In Abschnitt 2.2.1 wird das Functional Mock-up Interface (FMI) vorgestellt. Es definiert einen Austausch- und Kosimulationsstandard für HDG-Modelle. Dabei wird angenommen, dass die Modelle bestimmten Anforderungen genügen. Mit FMI kann die Heterogenität auf Werkzeug- und Notationsebene überwunden werden. In Abschnitt 2.2.2 wird das Ptolemy Framework vorgestellt. Dieses konzentriert sich auf die Kopplung von Modellen mit unterschiedlicher Semantik. Ptolemy definiert hierfür eine eigene Modellierungssprache und liefert eigene Werkzeuge.

### 2.2.1 Functional Mock-up Interface

Das Functional Mock-up Interface (FMI) (Blockwitz, 2012) ist ein Standard mit dem es möglich ist, die Heterogenität auf Ebene von Werkzeugen und Notationen zu überwinden. Die folgenden Ausführungen beziehen sich auf Version 2.0.1 des Standards (FMI, o. J.). Hierzu definiert der Standard das Austauschformat Functional Mock-up Unit (FMU). Dieses Format kann in den zwei Anwendungsfällen *Model-Exchange* und *Co-Simulation* genutzt werden.

FMUs können zum Austausch von HDG-Modellen zwischen Werkzeugen eingesetzt werden (***Model-Exchange***). Hierbei wird das Modell zunächst in einem Werkzeug modelliert und anschließend als FMU exportiert. Die FMU enthält eine Beschreibung der Gleichungen des Modells und macht diese über eine Schnittstelle zugreifbar. Diese FMU kann nun in ein anderes Werkzeug importiert und in ein Rahmenmodell eingebettet werden. Dieses Rahmenmodell kümmert sich im einfachsten Fall nur um das Lesen von Inputs. Die Simulation der Differenzialgleichungen erfolgt durch das importierende Werkzeug.

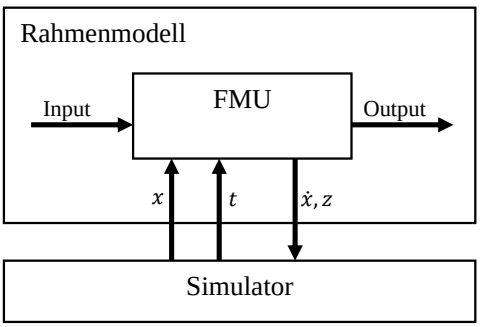



*Abbildung 2.7: FMU für Model Exchange in Anlehnung an den FMI-Standard* (FMI, o. J.).

Abbildung 2.7 gibt einen Überblick über diesen Modus. Der Simulator kann über die Schnittstellen der FMU nicht direkt den Zustand des Systems abfragen. Stattdessen übergibt

der Simulator einen Zustand $x$ und einen zugehörigen Zeitpunkt. Er erhält die Ableitung des Zustands zu diesem Zeitpunkt. Hiermit kann der Simulator nun die numerische Integration durchführen. Es ist auch möglich, Modelle unterschiedlicher Werkzeuge in ein Rahmenmodell zu integrieren. Hierzu werden die Modelle in einem Werkzeug geladen und mit den Modellierungsmitteln in diesem Werkzeug verbunden.

Eine zweite Variante ist die Kopplung von Simulatoren (***Co-Simulation***). In diesem Fall enthält die FMU nicht nur das Modell, sondern auch eine ausführbare Beschreibung des Simulators. Um FMUs zu koppeln, werden alle Simulatoren separat ausgeführt. Die Ein- und Ausgaben der FMUs werden an definierten Kommunikationszeitpunkten ausgetauscht. Wie genau dieser Austausch durchgeführt wird, wird mithilfe eines Masteralgorithmus definiert. Dieser wird durch den Modellierer definiert.

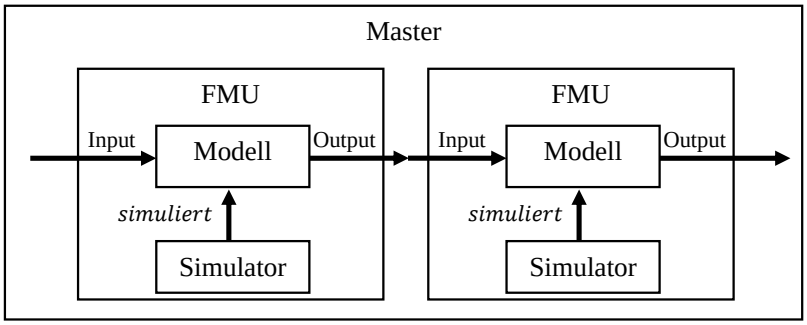



*Abbildung 2.8: FMU zur Co-Simulation in Anlehnung an den FMI-Standard* (FMI, o. J.).

In der Regel sind diese Kommunikationszeitpunkte gleichverteilt. Tatsächlich ist die Fähigkeit, mit nicht äquidistanten Intervallen zwischen den Kommunikationszeitpunkten umgehen zu können, keine Anforderung an eine FMU. Somit entspricht das Zeitverhalten der gekoppelten Simulation dem eines DT-Modells. Je nach Wahl des Masteralgorithmus ist auch die parallele Ausführung der FMUs möglich.

Dieser Aufbau erlaubt es, die FMUs mit ihren Simulatoren auf verschiedene Rechner zu verteilen. Hierdurch kann die Rechenleistung großer Cluster genutzt werden. Gleichzeitig gewährt dieser Aufbau große Freiheit bei der Gestaltung der Abläufe innerhalb der FMU. Die wesentliche Anforderung an Modell und Simulator ist die Möglichkeit, an den Kommunikationszeitpunkten Daten auszutauschen. Innerhalb der FMUs ist zwischen den Kommunikationszeitpunkten beispielsweise auch ein kontinuierliches Zeitverhalten des Zustands möglich.

### 2.2.2 Ptolemy Framework

Das Ptolemy Projekt beschäftigt sich mit dem Überbrücken von Heterogenität auf der semantischen Ebene. Die Erkenntnisse aus diesen Überlegungen sind in die Ptolemy II Software (Eker, 2003) eingegangen, welche als Open-Source-Projekt veröffentlicht wurde. Ptolemy verfügt über eine formale Semantik (Tripakis, 2013).

Ein zentrales Konzept des Ansatzes liegt darin, die Heterogenität zwischen Teilen eines Modells durch eine Hierarchie zu kapseln. Hierbei wird das modellierte System hierarchisch in Komponenten gegliedert. Diese werden Aktoren *(actors)* genannt. Ein Aktor tauscht Daten über sogenannte Ports mit seiner Umgebung aus. Diese Ports können als Sammlungen von Variablenbelegungen verstanden werden. Auch der Zustand eines Aktors entspricht der Belegung von Zustandsvariablen.

Ein atomarer Aktor spezifiziert sein Verhalten direkt. Das Verhalten eines Aktors kann sich aber auch aus der Komposition untergeordneter Aktoren ergeben. Atomare und zusammengesetzte Aktoren verfügen über dieselbe Schnittstelle. Diese setzt sich aus vier Funktionen zusammen:

- Eine *fire*-Funktion produziert aus dem Zustand des Aktors und seinen Eingaben Ausgaben. Hierbei wird der Zustand des Aktors nicht verändert.
- Dies geschieht in der *postfire*-Funktion. Diese produziert aus dem aktuellen Zustand und den Eingaben einen Folgezustand. Diese Aufteilung ist notwendig, um das Verhalten des Aktors mithilfe einer Fixpunktsemantik beschreiben zu können.
- Zudem liefert die *deadline*-Funktion den nächsten Zeitpunkt, zu dem der Aktor aktiviert werden muss.
- Schließlich kann mithilfe der *time*-Funktion der Zustand des Aktors basierend auf einer Zeit, die vergangen ist, geändert werden.

Aufgrund dieser gemeinsamen Schnittstelle können die Teile eines solchen zusammengesetzten Aktors wiederum atomar oder zusammengesetzt sein.

Wie genau die Teile eines zusammengesetzten Aktors interagieren, wird durch einen Direktor *(director)* definiert. Jeder zusammengesetzte Aktor kann einen eigenen Direktor definieren. Der Direktor ruft die Funktionen der untergeordneten Aktoren auf und tauscht die Daten zwischen ihnen aus, um die Funktionen des zusammengesetzten Aktors zu realisieren.

Die beschriebenen Modellarten entsprechen jeweils einem bestimmten Direktor. So wird etwa ein DE-Direktor die Ausgaben der Aktoren als Ereignisse auffassen und an nachgelagerte Aktoren weiterreichen. Die *deadline* Funktion liefert dann den Zeitpunkt des nächsten Ereignisses im System. Ein DT-Direktor gibt bei jeder *deadline* Anfrage den Zeitpunkt des

nächsten Simulationsschrittes zurück und nutzt die Fixpunktsemantik, um an diesem Zeitpunkt zunächst alle Ausgaben zu ermitteln und anschließend den Zustand zu ändern. Es entsteht ein Verhalten, welches einem DT-System entspricht. Auch DG- und HDG-Modelle können realisiert werden, indem entsprechende numerische Integrationsverfahren auf Basis der vier Funktionen definiert werden.

Diese Hierarchisierung sorgt dafür, dass innerhalb eines Aktors immer nur eine einzige Ausführungssemantik relevant ist. Hierdurch muss ein Modellierer, der einen dieser Aktoren entwickelt, nur diese eine Semantik kennen. Die eigentliche Integration zwischen den Modellarten erledigt das Konzept jedoch nicht. Hierfür werden eine Reihe von Werkzeugen in Form von Hilfs-Aktoren bereitgestellt.

Das folgende Beispiel einer Gasturbine stammt aus der Dokumentation von Ptolemy II (Ptolemy, o. J.) und verdeutlicht, wie die Hilfs-Aktoren die Kopplung von Modellen mit unterschiedlicher Semantik unterstützen können. Die Gasturbine wird in einem kleinen System eingesetzt. Ein Regler stellt die Treibstoffzufuhr der Turbine so ein, dass eine bestimmte Zielspannung von 110 Volt erreicht wird. Zudem wird zum Zeitpunkt t=15 eine Last angeschlossen. Überschreitet die Spannung einen kritischen Wert, wird die Last wieder getrennt. Beides wird als Ereignis modelliert. Entsprechend kommt der DE-Direktor zum Einsatz. Abbildung 2.9 zeigt das Modell des Systems.

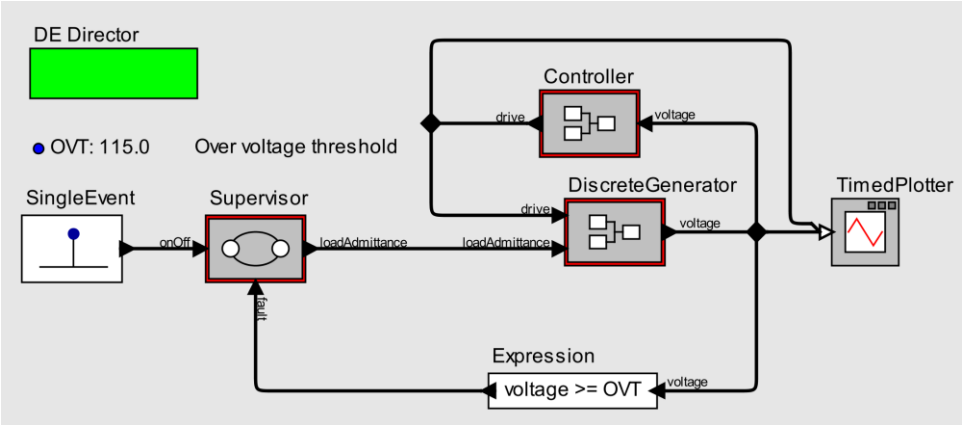



*Abbildung 2.9: DE Modell eines Systems aus einer Gasturbine, einer Last und einem Regler.*

Betrachten wir nun ein HDG-Modell der Gasturbine. Es ist in Abbildung 2.10 dargestellt. Die beiden Eingaben stehen für die Treibstoffzufuhr (*drive*) der Turbine sowie für die angeschlossene Last (*loadAdmittance*). Ein Limiter begrenzt die Treibstoffzufuhr, sodass diese nicht negativ werden kann. Das Verhalten des Systems kann durch die Differenzialgleichung $\dot{v}(t) = \frac{1}{T} * \big(d(t) - v(t)\big)$ beschrieben werden. Hierbei ist $d(t)$ die Ausgabe des Limiters und $v(t)$ die durch den Generator erzeugte Spannung. Der obere Teil des Modells beschreibt diese

Gleichung. Der untere Teil berechnet den Einfluss der Last auf die Ausgabespannung des Generators.

Um die differenzialgleichungsbasierte Formulierung simulieren zu können, wurde das Modell mit dem *continuous*-Direktor versehen. Dieser führt eine numerische Integration durch, wenn der gesamte, zusammengesetzte Aktor des Generators ausgeführt wird. Beide Modelle sind jedoch nicht direkt kompatibel. Die Treibstoffzufuhr und die Last werden als diskrete Ereignisse modelliert. Die Turbine benötigt für seinen Integrationsschritte jedoch kontinuierliche Werte. Zudem ist die Ausgabe des HDG-Modells kontinuierlich. Das übergeordnete System (Abbildung 2.9) übergibt diese Werte jedoch als diskrete Ereignisse an den Regler.

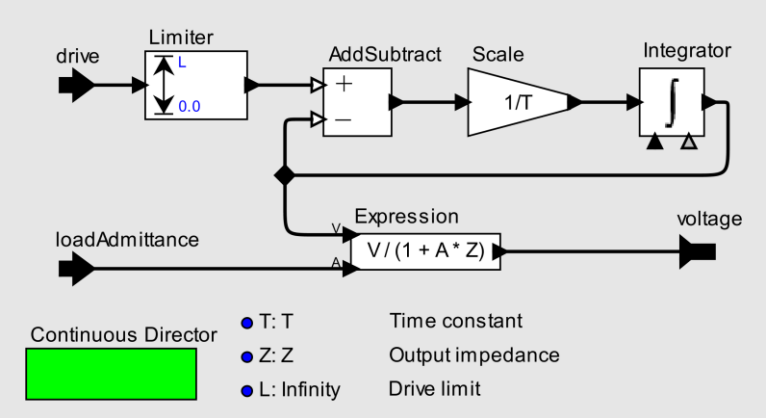



*Abbildung 2.10: Vereinfachtes DG Modell einer Gasturbine.*

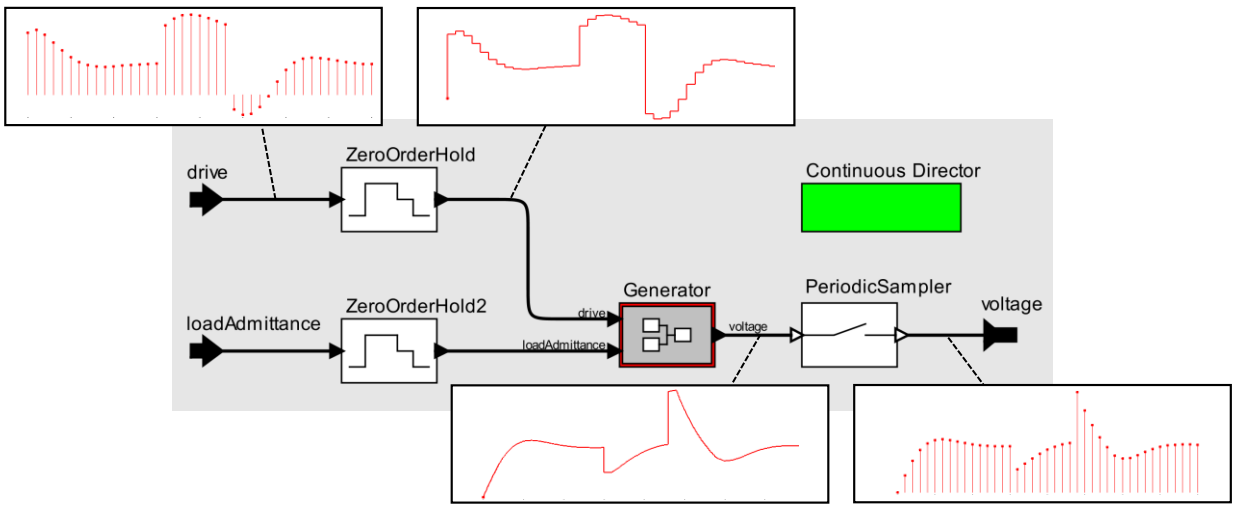



*Abbildung 2.11: Adapter zwischen DE Modell des Systems und DG Modell der Turbine.*

Aus diesem Grund muss das HDG-Modell der Turbine mit einem Adapter versehen werden. Abbildung 2.11 zeigt diesen Adapter. Auf der linken Seite werden zwei *ZeroOrderHold*

Aktoren eingesetzt, um die diskreten Eingaben in eine kontinuierliche Zeitachse zu übertragen. Dies geschieht, indem die Werte der Ereignisse gespeichert und solange als Konstanten ausgegeben werden, bis sie durch ein neues Ereignis überschrieben werden. Die Ausgabe des *ZeroOrderHold* enthält dann zwar immer noch Sprungstellen, dies ist jedoch für den Integrator (Abbildung 2.10) kein Problem. Die Ausgabe wird durch einen *PeriodicSampler* wieder in Ereignisse überführt. Hierbei wird immer nach einer vorgegebenen Periode ein Ereignis mit dem aktuellen Wert der kontinuierlichen Ausgabe des Generators erzeugt. Um die Modelle zu koppeln, wird das Modell der Gasturbine (Abbildung 2.10) in den Adapter (Abbildung 2.11) und dieser wiederum in das Modell des Systems (Abbildung 2.9) eingesetzt.

Dies ist ein Beispiel für die Kopplung zwischen DE- und DG-Modellen. Ptolemy verfügt über eine Bibliothek vergleichbarer Werkzeuge, welche die Kopplung verschiedenster Modellarten erlauben.

## 2.3 Wahrscheinlichkeitstheorie

Im weiteren Verlauf der Arbeit werden Zufallsvariablen von Wahrscheinlichkeitsräumen genutzt, um mit dem Informationsverlust einer Abstraktion umzugehen. Aus diesem Grund wird in diesem Abschnitt kurz auf die zugrunde liegende Wahrscheinlichkeitstheorie eingegangen. Die vorgestellte Notation folgt den Ausführungen von Georgii (Georgii, 2007).

Betrachten wir hierzu zunächst den Ergebnisraum $\Omega$. Hierbei handelt es sich um eine Menge, deren Elemente alle für den jeweiligen Anwendungsbereich relevanten Ausgänge eines Zufallsexperiments enthält.

Ein vielzitiertes Beispiel ist das Ergebnis eines Würfelwurfes. Hierbei ist lediglich die Anzahl der Augen, die nach dem Wurf auf der Oberseite des Würfels liegen relevant. Entsprechend ist hier $\Omega = \{1, ...,6\}$. Der Ergebnisraum ist keinesfalls auf diskrete oder auf endliche Mengen beschränkt. So können auch die reellen Zahlen oder reelle Vektoren als Ergebnisraum dienen: $\Omega \subset \mathbb{R}^n$

Oft interessieren bei einem Zufallsexperiment nicht Aussagen über einzelne Elemente des Raumes, sondern vielmehr Ereignisse, welche mehrere dieser Elemente zusammenfassen.

Betrachten wir erneut den Würfelwurf als Beispiel. Nehmen wir an, uns interessiert, ob die Zahl der Augen gerade ist. Der genaue Ausgang des Wurfs ist nicht relevant. Somit ergeben sich zwei unterschiedliche Ereignisse. Das Ereignis „Die Augenzahl ist gerade“ und das Ereignis „Die Augenzahl ist ungerade“.

Wir wollen solche Ereignisse mit Wahrscheinlichkeiten belegen können. Zu diesem Zweck definieren wir ein Mengensystem, welches alle Teilmengen des Ergebnisraumes enthält, über die wir Aussagen treffen können. Im Beispiel des Würfels (und tatsächlich für alle endlichen Ergebnisräume) kann hierfür die Potenzmenge des Ereignisraumes $\mathbb{P}(\Omega)$ genutzt werden. Ist der Ereignisraum jedoch nicht endlich, wie etwa im Fall $\Omega \subset \mathbb{R}^n$, ist $\mathbb{P}(\Omega)$ „zu groß“, um darauf später eine Abbildung zur Wahrscheinlichkeitsbewertung zu definieren (Beweis siehe (Georgii, 2007, S. 9ff)).

Deshalb sei $\Sigma \subset \mathbb{P}(\Omega)$, die sogenannte $\sigma$-Algebra. Für den Ergebnisraum $\Omega \subset \mathbb{R}^n$ ist die sogenannte borelsche $\sigma$-Algebra $\mathfrak{B}^n$ geeignet. Sie enthält alle Teilmengen von $\mathbb{R}^n$, denen man nativ ein Volumen zuordnen würde, ist aber gerade noch nicht „zu groß“ um allen Elementen eine Wahrscheinlichkeit zuweisen zu können.

Das Paar $(\Omega, \Sigma)$ wird als Ereignisraum aus dem Ergebnisraum $\Omega$ und der $\sigma$-Algebra $\Sigma$ bezeichnet.

Ein Wahrscheinlichkeitsmaß ist eine Funktion $P: \Sigma \to [0,1]$ die jedem Ereignis aus $\Sigma$ einen Wahrscheinlichkeitswert zwischen 0 und 1 zuweist. Das Tripple $(\Omega, \Sigma, \mathrm{P})$ wird Wahrscheinlichkeitsraum genannt.

Betrachten wir nun zudem den Ereignisraum $(\Omega', \Sigma')$ und die Funktion $X: \Omega \to \Omega'$. X ist eine Abbildung von einem Ergebnisraum $\Omega$ in einen Ergebnisraum $\Omega'$. Hiermit ist es möglich beispielsweise Ergebnissen Zahlenwerte zuzuordnen. Typischerweise wird damit der betrachtetet Modellausschnitt auf einen relevanten Aspekt beschränkt. Eine derartige Funktion wird Zufallsvariable genannt.

Betrachten wir als Beispiel den n-fachen Würfelwurf. Dieser kann mit dem Ergebnisraum $\Omega = \{1, \dots, 6\}^n$ mit den Elementen $\omega = (\omega_1, \dots, \omega_n)$ beschrieben werden. Nun ist es möglich, dass nicht die Werte der einzelnen Würfel, sondern deren Summe interessiert. Dies kann durch eine Zufallsvariable $X: \Omega \to \{n, \dots, n * 6\}\ mit\ X(\omega) = \sum_{i=1}^{n} \omega_i$ beschrieben werden.

Eine zentrale Forderung an Zufallsvariablen ist, dass das Urbild jedes Ereignisses aus $\Sigma'$ in $\Sigma$ liegt.

$A' \in \Sigma' \Rightarrow X^{-1}(A') \in \Sigma.$

Intuitiv bedeutet dies, dass sich alle Ereignisse von X auf Ereignisse von $\Sigma$ zurückführen lassen. Im Beispiel des 2-fachen Würfelwurfes kann etwa das Ereignis $\{4\} \in \Sigma'$ („die Summe der Augen beider Würfel ist 4“) auf $\{(1,3), (2,2), (3,1)\} \in \Sigma$ zurückgeführt werden. Hierbei ist zu beachten, dass hier nicht Ergebnisse, also z.B. $4 \in \Omega'$ und $(1,3) \in \Omega$ , sondern Mengen von

Ergebnissen aus den beiden $\sigma$-Algebren betrachtet werden. Hierdurch ist das Urbild eindeutig definiert (als genau eine Menge aus dem Mengensystem Σ).

Durch diese Anforderung ist das Wahrscheinlichkeitsmaß $P_X(A') = P\left(X^{-1}(A')\right)$ für die Verteilung von X eindeutig definiert. Somit entsteht der Wahrscheinlichkeitsraum $(\Omega', \Sigma', \mathrm{P_X})$.

Ein weiteres wichtiges Konzept der Wahrscheinlichkeitstheorie ist die Abhängigkeit von Ereignissen. Nehmen wir an, wir ziehen aus einer Urne mit schwarzen und weißen Kugeln blind eine schwarze Kugel. Wenn wir nun, ohne Zurücklegen, eine weitere Kugel ziehen, würden wir eine schwarze Kugel für weniger wahrscheinlich halten als bei dem ersten Zug. Die Ereignisse „erste Kugel schwarz“ und „zweite Kugel schwarz“ sind abhängig voneinander.

Dies kann mit einer bedingten Wahrscheinlichkeit modelliert werden. Seien $A, B \in \Sigma$ Ereignisse, dann ist

$$P(B|A) := \frac{P(A \cap B)}{P(A)}$$

die bedingte Wahrscheinlichkeit von B unter der Bedingung A, bezügliche des Wahrscheinlichkeitsmaßes P.

Im Laufe dieser Arbeit wird wiederholt das Wahrscheinlichkeitsmaß $N_{m,v}$ auf dem Ereignisraum $(\mathbb{R}, \mathfrak{B})$ Verwendung finden. Hierbei handelt es sich um die Normalverteilung. Die Dichtefunktion ist durch

$$\phi_{m,v}(x) = \frac{1}{\sqrt{2\pi v}} e^{-\frac{(x-m)^2}{2v}}$$

gegeben. Der Parameter $m$ entspricht dem Erwartungswert der Verteilung und $v$ entspricht ihrer Varianz.

Folgt eine Zufallsvariable einer Verteilung wird im weiteren Verlauf das Symbol ~ verwendet. Ein Beispiel hierfür ist die normalverteilte Zufallsvariable $X$ mit Erwartungswert $m$ und Varianz $v$.

$$X \sim N_{m,n}$$

## 2.4 Simulation im Entwicklungsprozess

In diesem Abschnitt wird beleuchtet, wie Simulation innerhalb des Entwicklungsprozesses komplexer Systeme eingesetzt werden kann. Dabei werden wir uns am Entwicklungsprozess

mechatronischer Systeme orientieren. Dieser wird in der VDI-Richtline 2206 – „Entwicklungsmethodik für mechatronische Systeme“ (VDI, 2003) beschrieben.

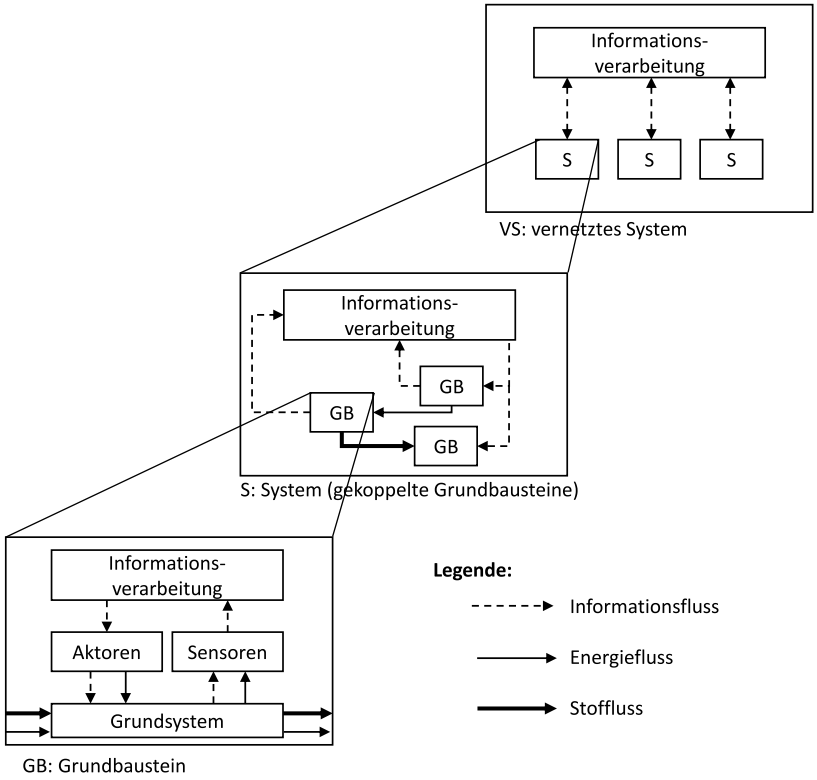



*Abbildung 2.12 Hierarchie mechatronischer Systeme, nach Lückel* (Lückel, 2000).

Dabei versteht diese Richtlinie mechatronische Systeme als Zusammenfassung von vier wesentlichen Strukturelementen. Zunächst ist hier das **Grundsystem** zu nennen. Hierbei handelt es ich um ein physikalisches System wie beispielsweise einen Otto-Motor. **Sensoren** ermöglichen das Erfassen des Zustands dieses Grundsystems. Dieser Zustand ist die Eingabe in eine **Informationsverarbeitung** (z.B. ein Motor Steuergerät). Die Informationsverarbeitung bestimmt, ob und wie auf das Grundsystem eingewirkt werden soll, um den Zustand zu verändern. Diese Einwirkung wird mittels **Aktoren** (nicht zu verwechseln mit dem Aktor (*actor*) des Ptolemy-Frameworks, siehe 2.2.2) an dem Grundsystem durchgeführt. Diese Elemente und ihre Verbindungen finden sich in Abbildung 2.12 unten links.

Das Grundsystem kann wiederum Teil eines komplexeren physikalischen Systems sein. In diesem Fall wird das mechatronische System aus der Integration mehrerer derartiger **Grundbausteine** aufgefasst. Abbildung 2.12 zeigt diese Hierarchisierung. Ein Beispiel für ein solches **System** aus verkoppelten Grundbausteinen ist ein Fahrzeug. Auf einer dritten Ebene können wiederum mehrerer solcher Systeme zu einem sogenannten **vernetzten System** verbunden sein. Ein Beispiel für ein solches vernetztes System aus der Richtlinie ist ein Kreuzungsmanagement, in dem eine Vielzahl von Fahrzeugen interagieren.

Aus der engen Interaktion der Elemente auf den verschiedenen Ebenen und auf Grund der Wechselwirkungen zwischen den Elementen, ergibt sich die Notwendigkeit, bei der Entwicklung dieser Systeme eine ganzheitliche Perspektive zu erreichen.

> *„Die wachsende Integration von Funktionen, Wirkprinzipien/Lösungselementen und Technologien führt zu Wechselwirkungen, die so früh wie möglich berücksichtigt werden müssen. Das Vorgehen, getrennt entwickelte und optimierte Baugruppen zu einem Gesamtsystem zusammenzufügen (bottom-up-design), ist nicht mehr ausreichend. Es sind iterative Vorgehensschritte nötig, um zunächst Kenntnisse der Grobstruktur zu erlangen und dann durch schrittweise Verfeinerung die Strukturelemente genauer zu spezifizieren“ (top-down-design).“* (VDI, 2003, S. 13)

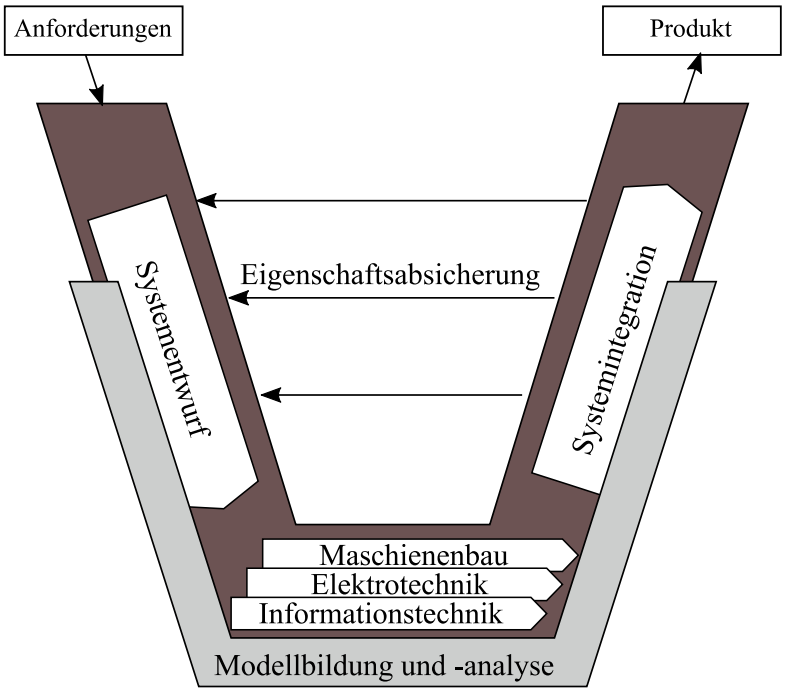



*Abbildung 2.13: V-Modell mechatronischer Systeme. Nach Richtlinie 2206* (VDI, 2003, S. 16).

Aus diesem Grund schlägt die Richtlinie ein maßgeschneidertes V-Modell für die Entwicklung mechatronischer Systeme vor. Abbildung 2.13 zeigt dieses V-Modell.Das übergeordnete Vorgehen besteht dabei darin, zunächst das angestrebte System in immer kleinere Einheiten und deren Schnittstellen zu zergliedern und abschließend in domänenspezifischen Entwürfen auszuspezifizieren. Dies geschieht auf der linken Seite des V-Modells. Auf der rechten Seite wird das System in immer größere Einheiten integriert und die Eigenschaften dieser Einheiten gegenüber der Spezifikation abgesichert (Verifikation).

Beide Seiten des V-Modells werden von Modellbildung und -analyse begleitet. Modellierung und Simulation, wie sie bereits in Abschnitt 2.1 diskutiert wurden, findet innerhalb dieses Vorgehensbausteins statt.

Nehmen wir an, der Entwicklungsgegenstand ist ein System gemäß der Hierarchie in Abbildung 2.12. Dann fließen **Anforderungen** an dieses System als Eingabe in den Entwicklungsprozess.

Die erste Aktivität im Prozess ist nun der **Systementwurf**. Hierbei wird definiert, aus welchen Grundbausteinen sich das System zusammensetzen soll und wie die Schnittstellen zwischen ihnen aussehen. Neben der Definition von Schnittstellen der beteiligten Informationsverarbeitung müssen hier auch Energie- und Stoffflüsse zwischen den jeweiligen Grundsystemen ausgestaltet werden. Schon von Beginn an werden dabei Alternativentwürfe mithilfe von Simulation gegenübergestellt und bewertet.

Die Entwicklung der Grundbausteine erfolgt meist getrennt nach den beteiligten Domänen (VDI, 2003). Dabei entstehen **domänenspezifische (Fein-)Entwürfe** der Disziplinen Maschinenbau, Elektronik und Informationstechnik. Die Erstellung dieser Feinentwürfe erfolgt gemäß den in den Domänen etablierten Methoden. Simulation erlaubt hier die Gegenüberstellung und Bewertung verschiedener Feinentwürfe für die Grundbausteine.

Im beschriebenen Fall von zwei Ebenen (System und Grundbausteine) wird also das System zunächst in Teilsysteme (Grundbausteine) und anschließend in die Domänen unterteilt entwickelt.

Wie genau hierbei Modellierung und Simulation zur Unterstützung eingesetzt werden können, wird in der Richtlinie im Rahmen der Methode des **modellbasierten Systementwurfes** vorgeschlagen. Hierbei werden für jedes Teilsystem Modelle erstellt. Der Detailgrad der Modelle ist entsprechend der untersuchten Fragestellung zu wählen:

> *„Je nach Fragestellung variiert die Modellierungstiefe in Hinblick auf die Berücksichtigung bestimmter physikalischer Effekte. So genügt in bestimmten Fällen die Modellierung eines aus mehreren Komponenten bestehenden mechanischen Systems (z.B. Kfz) als eine Punktmasse, während in anderen Fällen komplexe Mehrkörpermodelle oder auch Finite-Elemente-Modelle aufgebaut werden müssen.“* (VDI, 2003, S. 27)

Diese Modelle sollen zudem entlang des Entwicklungsprozesses weiter verfeinert werden:

> *Idealerweise bauen Modelle späterer Entwicklungsphasen auf Modellen früherer Phasen auf. Ausführbare Spezifikationen, die in frühen Entwicklungsphasen grob die Funktion eines Systems beschreiben, können*

> *beispielsweise für den Aufbau von detaillierteren Verhaltensmodellen genutzt werden. […] Diese so genannte Durchgängigkeit sollte sinnvollerweise über alle Entwicklungsphasen bis hin zum endgültigen System erhalten bleiben.* (VDI, 2003, S. 26)

Das Vorgehen zum modellbasierten Systementwurf *(VDI, 2003)* für ein bestimmtes Teilsystem beginnt mit einer **Zielformulierung**. Hierbei wird im Wesentlichen ein Untersuchungsziel und somit letztlich der experimentelle Rahmen aus Abbildung 2.1 definiert. Dabei setzt sich das Quellsystem aus Abbildung 2.1 immer aus der Kombination eines physikalischen Grundsystems (vgl. Abbildung 2.12) mit einer Informationsverarbeitung sowie Sensoren und Aktoren zusammen. Wir werden Informationsverarbeitung, Sensoren und Aktoren im weiteren Verlauf als System bezeichnen.

Zunächst wird im Rahmen der **Modellbildung** ein Modell des Grundsystems erstellt. Dieses wird in der Richtlinie als Ersatzmodell bezeichnet. Anschließend wird dieses Ersatzmodell im Rahmen der **Modellanalyse** auf für den Entwurf des Systems wichtige Eigenschaften hin untersucht. Hierzu muss das Modell zunächst in eine ausführbare Form gebracht und durch einen Simulator simuliert werden. Schließlich wird in der **Systemsynthese** ein Entwurfsvorschlag für das System erstellt. Dieser Vorschlag kann beispielsweise ein ausführbares Modell des Systems verbunden mit dem Ersatzmodell sein. In der **Systemanalyse** wird nun dieser Lösungsvorschlag bewertet. Bei nicht Erfüllung der Anforderungen sind Rücksprünge zur Synthese möglich.

Sind die Feinentwürfe abgeschlossen folgt auf dem rechten Schenkel des V-Modells die **Systemintegration**. Hierbei werden die domänenspezifischen Entwürfe zu Teilsystemen und diese Teilsysteme schließlich zu einem System integriert. In dieser Integration sind jedoch unter Umständen Anpassung aufgrund von Inkompatibilitäten notwendig. Die Richtlinie beschreibt dieses Problem explizit:

> *„Um einen hohen Integrationsgrad zu erreichen, sind bereits beim Systementwurf die Wirkprinzipien und Lösungselemente unter Berücksichtigung der Nutz- und Störfunktionen auf Kompatibilität zu überprüfen und die Schnittstellen für die spätere Integration auszubilden (Grobdimensionierung).*
>
> *Während der Feindimensionierung in den involvierten Fachdisziplinen (domänenspezifischer Entwurf) kann es jedoch zu Veränderungen der Wirkstruktur (durch Funktionsintegration, Funktionstrennung etc.) und der Baustruktur kommen. Mögliche Inkompatibilitäten müssen deshalb bei der Systemintegration erkannt und eliminiert werden.“* (VDI, 2003, S. 20)

Um für die einzelnen Integrationsstufen zu prüfen, ob die im Entwurf geforderten Eigenschaften eingehalten werden, wird jeweils eine **Eigenschaftsabsicherung** durchgeführt. Diese Absicherung kann, je nach Sicherheitsanforderungen an das Produkt, in Form eines physikalischen Prototyps, eines teilweise virtuellen Prototyps (z.B. im Sinne von Hardware-in-the-Loop-Simulation) oder vollständig virtuell (im Sinne der Simulation eines virtuellen Prototyps) erfolgen. Ergebnis des Entwicklungsprozesses ist dann ein integriertes und abgesichertes **Produkt**.

Innerhalb des dargestellten Entwicklungsprozesses wird also zunächst ein Systementwurf erstellt. Dieser legt einen groben Verhaltensrahmen für das entwickelte System fest. Zudem definiert der Systementwurf Teilsysteme und deren Schnittstellen. Bei der Erstellung dieses Systementwurfs werden bereits früh ausführbare Modelle des Systems erstellt, um Alternativen bewerten zu können. Hierbei wird das System als Ganzes untersucht.

Die Ausgestaltung von Feinentwürfen erfolgt dann mit einem eingeschränkten Blickfeld auf einzelne Teilsysteme. Auch diese Teilsysteme werden modelliert und simuliert, um bewertet werden zu können. Dabei liegt der Fokus naturgemäß auf dem untersuchten Teilsystem. Ob sich diese Teile auch tatsächlich zu dem gewünschten System zusammenfügen, kann unter Umständen erst bei einer Systemintegration bewertet werden. Im schlimmsten Fall erst bei dem Aufbau eines integrierten, physikalischen Prototyps.

Dieses prinzipielle Problem ergibt sich aus der notwendigen und sinnvollen Aufteilung des Systems in kleinerer Betrachtungseinheiten. Der Entwicklungsprozess mechatronischer Systeme ist dabei nur ein Beispiel.

Es existieren verschiedenen Ansätze, um die Modelle, welche im Laufe der Entwicklung erstellt werden, zu nutzen, um bereits vor der Integration des Systems zu einer ganzheitlicheren Bewertung zu gelangen. Die bereits beschriebene Kopplung von Modellen (Abschnitt 2.2) ist ein Beispiel dafür. In Kapitel 3 werden diese Lösungsansätze anhand eines Beispiels diskutiert.

# 3 Beispiel: Entwicklung einer Lieferkette

Im folgenden Abschnitt wird das fiktive Szenario der Entwicklung einer Lieferkette vorgestellt. Das Ziel dieses Beispiels ist es, einen leicht verständlichen Kontext für die Kopplung von Simulationen unterschiedlicher Abstraktionsstufen zu liefern. Hierdurch wird auch das zentrale Problem, dass hierbei entsteht anschaulich dargestellt.

Die fiktive Lieferkette soll eingehende Waren lagern, bis eine entsprechende Bestellung eingeht. Anschließend müssen die Waren kommissioniert und nach einer abschließenden Qualitätssicherung versandt werden.

Wie im Abschnitt 2.4 dargestellt, wird hierbei zunächst ein Grobentwurf der gesamten Kette erstellt und simulativ bewertet (Abschnitte 3.1, 3.2 und 3.3). Anschließend werden die einzelnen Standorte der Lieferkette in Feinentwürfen ausgestaltet und ebenfalls modelliert sowie simuliert (Abschnitt 3.4, 3.5 und 3.6).

Dies ist der Ausgangspunkt für das Ziel, die Integration des Feinentwurfes in den Kontext des Grobentwurfes bereits vorab, vor der eigentlichen Integration, ebenfalls simulativ zu bewerten (Abschnitt 3.7). Anschließend werden drei prinzipielle Lösungsansätze zum Erreichen dieses Ziels diskutiert (Abschnitte 3.8, 3.9 und 3.10).

## 3.1 Anforderung an die Lieferkette

Im Rahmen des Beispiels wird eine ausgewählte Anforderung an die Lieferkette betrachtet. Dies wir im Folgenden dargestellt.

Das Versandhaus garantiert seinen Kunden, dass von der Bestellung bis zur Lieferung höchstens 24 Stunden vergehen und zahlt eine Entschädigung, falls dies nicht eingehalten wird. Der letzte Schritt der Lieferung wird von einem Logistikdienstleister erbracht. Dieser garantiert wiederum eine Lieferung innerhalb von 15 Stunden. Entsprechend bleiben der Lieferkette nach Eingang der Lieferung 9 Stunden, bis die Sendung an diesen Dienstleister übergeben werden muss. Diese Zeitspanne wird Durchlaufzeit genannt. Daraus ergibt sich die folgende Anforderung.

> *A1: Die Durchlaufzeit muss bei 90% der Lieferungen weniger als 540 Minuten*
> *(9 Stunden) betragen.*

Die Lieferkette muss so gestaltet werden, dass diese Anforderung erfüllt wird.

## 3.2 Modell der Lieferkette

Ein Planer wird beauftragt einen Grobentwurf für die Lieferkette anzufertigen. Um evaluieren zu können, ob dieser Entwurf die Anforderung A1 erfüllt, wird ein Modell der Lieferkette erstellt, dass anschließend simuliert wird.

Die Lieferkette soll aus drei Standorten bestehen. Der erste Standort ist ein Lager. Hier verweilen die Waren, bis sie bestellt werden. In dem zweiten Standort, einer Kommissionierung, werden sie zu Sendungen zusammengefasst. Im dritten Standort, der Qualitätssicherung, werden die kommissionierten Sendungen einer Prüfung unterzogen. Hierbei sollen mögliche Mängel vor dem Versand an den Kunden erkannt werden. Es ist beispielsweise möglich, dass die Umverpackung auf dem Weg zwischen den Standorten Schaden genommen hat. In solchen Fällen werden die Waren wieder ausgepackt und zurück an das Lager gesendet. Das Modell bildet die Standorte sowie die Logistikwege zwischen ihnen ab und erlaubt Durchlaufzeiten der Sendungen zu bestimmen.

Das Modell beschreibt die zeitabhängige Interaktion zwischen den Standorten. Eine Interaktion kann dabei als das Ändern von Variablenbelegungen aufgefasst werden. Verlässt eine Ware einen Standort, wird ein entsprechender Warenausgangspuffer beschrieben. Ein nachfolgender Standort kann diese Ware dann z.B. in einen Puffer kopieren und die entsprechende Variable des Ausgangspuffer auf null setzen. Auch das Zusammenspiel der Lieferkette mit seiner Umgebung wird auf diese Weise modelliert. Die Lieferkette verfügt über einen Eingang für Waren und einen für Bestellungen. Auf diese Weise entsteht ein DT Modell. Ein Zeitschritt des Modells entspricht einer Minute.

Abbildung 3.1 stellt das Modell schematisch dar. Zudem wir in der Abbildung ein Zustand des Modells dargestellt. Hierbei steht jeder Pfeil für eine Variable, welche für die Interaktion zwischen zwei Standorten genutzt wird. Der Pfeil ist mit der Belegung dieser Variable im betrachteten Zustand beschriftet.

Die Waren werden als Pakete modelliert. Die Pakete haben eine Identität. Im Modell ist diese durch eine explizite ID realisiert. Im Folgenden wird jedoch der Name des Paketes diese Funktion erfüllen, um die Darstellung einfach zu halten. Die Pakete haben einen Adressaten sowie ein Volumen. Im Folgenden werden Sie als Tupel in $R^2$ dargestellt. Beispiele für drei Pakete mit demselben Adressaten $(7)$ und unterschiedlichen Volumina $(10, 12, 20)$ sind $\mathrm{p1} = (7{,}10)$, $\mathrm{p2} = (7{,}12)$ und $\mathrm{p3} = (7{,}20)$. Hierbei stehen p1, p2 und p3 für die Namen der Pakete.

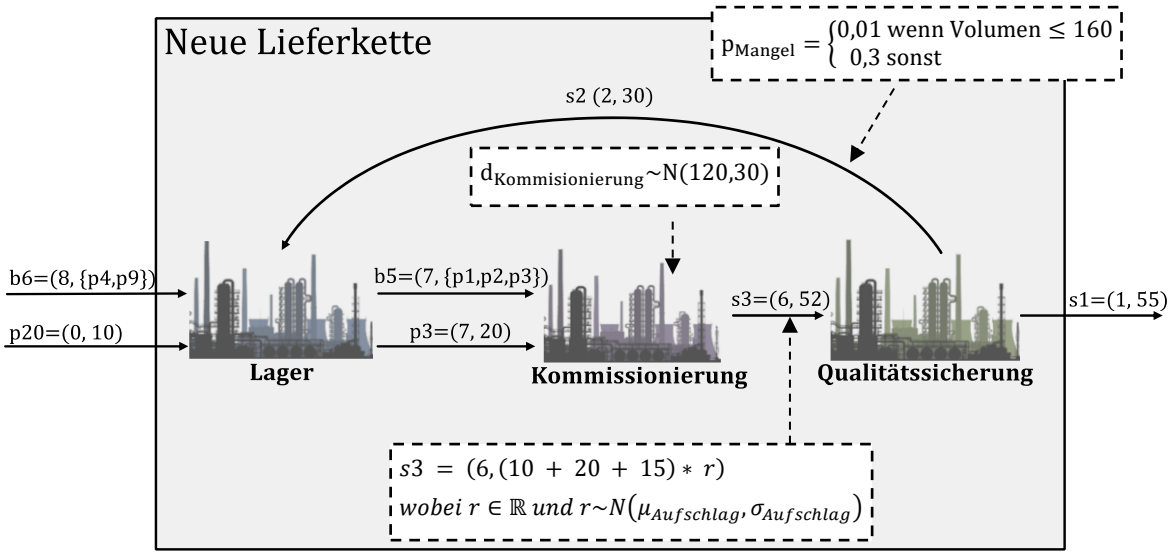



*Abbildung 3.1: Schematische Darstellung des Modells der Lieferkette.*

Wenn ein Paket die Lieferkette betritt, ist der Adressat zunächst 0. Im Lager werden eingehende Pakete gespeichert, bis eine entsprechende Bestellung eingeht. Bestellungen haben ebenfalls einen Adressaten und eine Liste von Paket-Identitäten. Ein Beispiel für eine solche Bestellung ist $\mathrm{b5} = (7, \{\mathrm{p1}, \mathrm{p2}, \mathrm{p3}\})$. Als Reaktion auf die Bestellung versieht das Lager die bestellten Pakete mit dem Adressaten, der in der Bestellung angegeben ist. Anschließend lagert es diese Pakete aus und sendet sie an die Kommissionierung. Das Auslagern eines Paketes erfolgt mit einer zufälligen Verzögerung. Mit dem letzten Paket einer Bestellung wird auch die Bestellung an die Kommissionierung weitergegeben. In der Abbildung 3.1 wird gerade die Bestellung $b5$ an die Kommissionierung übertragen. Zudem verlässt noch Paket $p3$ das Lager.

In der Kommissionierung werden die Pakete gepuffert. Beim Eintreffen der zugehörigen Bestellung werden sie zu einem größeren Paket zusammengefasst (kommissioniert). Im Folgenden werden solche zusammengefassten Pakete als Sendungen bezeichnet. Im Modell werden Sendungen und Pakete auf dieselbe Weise abgebildet. Der Adressat der Sendung wird von den Paketen übernommen. Das Volumen wird durch die Summe der Pakete plus einen normalverteilten, prozentualen Aufschlag bestimmt.

Bei der Kommissionierung der Pakete

$$\mathrm{p1} = (7{,}10), \mathrm{p2} = (7{,}12) \text{ und } \mathrm{p3} = (7{,}20)$$

entsteht die Sendung

$$\mathrm{s5} = (7, (10 + 12 + 20) * \mathrm{r}) \text{ wobei } \mathrm{r} \in \mathbb{R} \text{ und } \mathrm{r} \sim \mathrm{N}(\mu_{\mathrm{Aufschlag}}, \sigma_{\mathrm{Aufschlag}}).$$

Die Parameter $\mu_{\text{Aufschlag}}$ und $\sigma_{\text{Aufschlag}}$ dieser Verteilung werden aus Erfahrungswerten bei bestehenden Standorten geschätzt. In Abbildung 3.1 verlässt $s5$ jedoch noch nicht die Kommissionierung, da diese noch auf das letzte Paket für die Sendung wartet. Stattdessen ist die Sendung s3 auf dem Weg zum nächsten Standort. Sie wurde im vorrangegangen Zeitschritt kommissioniert.

Auch die Durchlaufzeit der Kommissionierung wird mit einer Normalverteilung modelliert. Im späteren Verlauf der Entwicklung soll sie so entworfen werden, dass die Lieferkette insgesamt ihre Anforderung (siehe Abschnitt 3.1) erfüllt. Eine Konfiguration, welche die Anforderung erfüllt, ist die folgende Verteilung.

$$d_{\text{Kommisionierung}} \sim N(120,20)$$

Bei der Qualitätssicherung werden zufällig einige der Sendungen als defekt ausgeschleust. Der Planer weiß, dass die Wahrscheinlichkeit dafür, dass auf dem Transport Defekte auftreten, von der Größe der Sendung abhängt. Die folgende Formel kommt zum Einsatz.

$$p_{\text{Mangel}} = \begin{cases} 0{,}01 \text{ wenn Volumen} \leq 160 \\ 0{,}3 \text{ sonst} \end{cases}$$

Auch diese Formel wurde durch die Analyse bestehender Anlagen ermittelt.

Sendungen ohne einen Defekt verlassen die Lieferkette durch einen Ausgang. Wenn eine Bestellung die Lieferkette betritt, wird die Zeit genommen. Dies geschieht auch, wenn eine Sendung die Lieferkette verlässt. Über den Adressaten können Bestellung und Sendung in Verbindung gebracht werden. Die Durchlaufzeit einer Bestellung ist die Differenz dieser Zeiten.

## 3.3 Inputs und Simulation der Lieferkette

Um das Modell simulieren zu können, muss der Planer einen experimentellen Rahmen ausgestalten. Deshalb definiert er die Inputs für die beiden Eingänge. Hierfür nutzt er Daten, die bei einer bestehenden Lieferkette des Unternehmens erhoben wurden. Für die eintreffenden Pakete überträgt er sowohl das Volumen als auch deren zeitlich Abfolge in eine Sequenz mit insgesamt 1000 Wareneingänge. Abbildung 3.2 zeigt ein Histogramm zu den Volumina der Pakete in der Sequenz.

Auf Basis derselben Daten erstellt der Planer zudem eine Sequenz von Bestellungen. Diese bestehen jeweils aus bis zu 7 Paketen. Die Zeitpunkte der Bestellungen werden ebenfalls aus den Daten entnommen. Insgesamt enthält die Sequenz 200 Bestellungen.

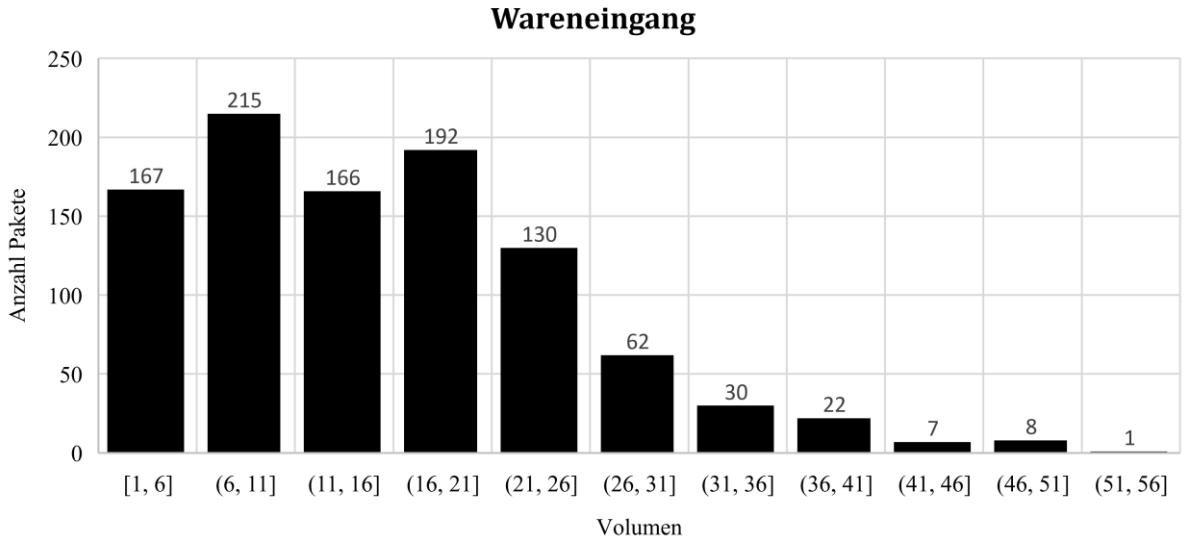



*Abbildung 3.2: Histogramm zu den Volumina der Pakete, die in der Lieferkette eintreffen.*

Abbildung 3.3 zeigt die Umsetzung des Modells in Ptolomy II. Dabei wird der Discret-Time-Direktor verwendet, welcher eine DT-Modell realisiert.

Der Planer simuliert das Modell und wertet die Durchlaufzeit der Bestellungen aus. Er stellt fest, dass die Anforderung A1 eingehalten wird. Nun kann mit der Ausgestaltung der Standorte begonnen werden. Im Rahmen des Beispiels wird jedoch nur die Kommissionierung beschrieben.

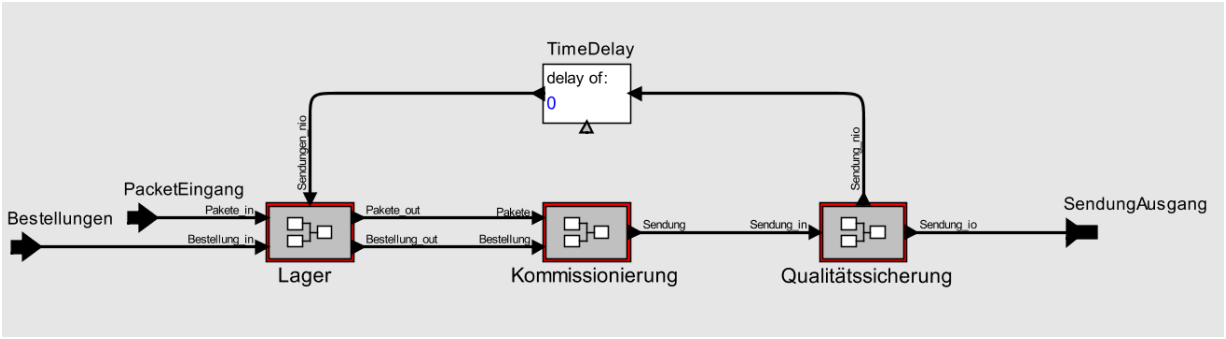



*Abbildung 3.3: Umsetzung des Modells der Lieferkette in Ptolemy II.*

## 3.4 Anforderungen an die Kommissionierung

Um die Anforderung A1 zu erfüllen, müssen die Teile der Lieferkette wiederum Anforderungen an ihre Durchlaufzeit einhalten. Für die Kommissionierung werden folgende Anforderungen festgeschrieben:

> *A1.1: Bei 90% der Sendungen muss die Durchlaufzeit in der Kommissionierung kleiner als 180 Minuten (3 Stunden) sein.*

Hierbei ist die Durchlaufzeit der Kommissionierung als die Zeit vom Eintreffen des letzten Teiles einer Sendung bis zur Ausgabe der kommissionierten Sendung definiert. Die Anforderungen für die anderen Standorte werden analog definiert, sollen aber für dieses Beispiel nicht weiter betrachtet werden.

## 3.5 Modell der Kommissionierung

Der Grobentwurf steckt nur einen groben Spezifikationsrahmen für die einzelnen Standorte ab. Die Simulation der Lieferkette legt nahe, dass die Anforderung A1 erfüllt werden kann, wenn die Standorte diesen Rahmen einhalten. Wie die Standorte jedoch konkret aussehen müssen, um diesen Rahmen einzuhalten, ist noch offen. Genau hier setzen die Feinentwürfe der Standorte an. Sie gestallten den Rahmen aus und beschreiben konkrete Realisierungen der Standorte. Ein Beispiel hierfür ist der Feinentwurf der Kommissionierung.

Die Kommissionierung muss die Anforderung A1.1 erfüllen, damit die Lieferkette insgesamt die Anforderung A1 erfüllen kann. Ob dies dem Feinentwurf des Planers gelingt, ist jedoch unklar. Um dies mit einer Simulation evaluieren zu können, modelliert der Planer den Feinentwurf der Kommissionierung.

Die eingehenden Pakete werden zunächst in zwei unterschiedlichen Puffern abgelegt. Um den Lagerraum der Puffer optimal nutzen zu können, gibt es zwei verschiedene Puffer. Der Puffer A hat Fächer, die sich gut für längliche, stangenartige Pakete eigenen. Flache, plattenartige Pakete würden hier Platz verschwenden. Der Puffer B hat Fächer, die sich gut für flache Pakete eignen. Aus beiden Puffern werden die Pakete in eine Kommissionieranlage gegeben. Hier kommt eine neuartiges Kommissionierverfahren zum Einsatz, welches die Abmaße des Ausgabepaketes zu minimieren versucht.

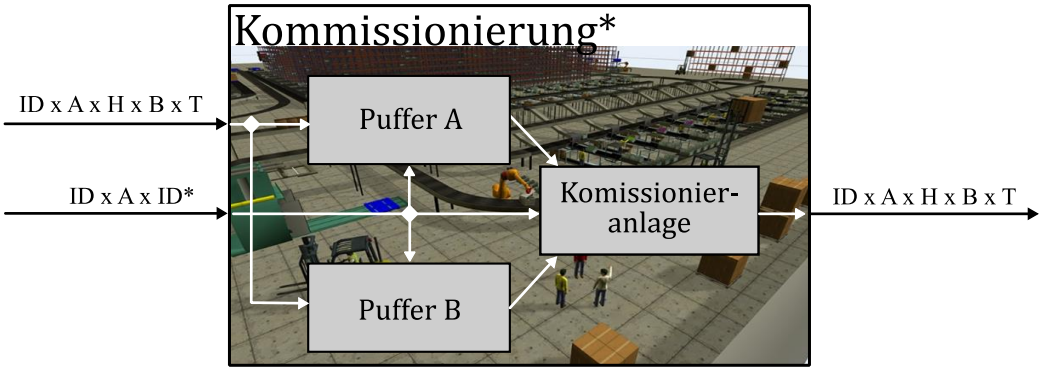



*Abbildung 3.4: Schematische Darstellung des Modells der Kommissionierung.*

Abbildung 3.4 gibt einen schematischen Überblick über das Modell. Im Folgenden wird dieses Modell als „Kommissionierung*“ bezeichnet.

Für das Modell sind die Abmaße der Pakete von entscheidendem Interesse. Die beiden Puffer arbeiten besser, wenn sie mit korrekt geformten Paketen bestückt werden und auch die Kommissionierung arbeitet auf Basis von dreidimensionalen Paketen. Entsprechend werden die Pakete durch Adressaten, Höhe, Breite und Tiefe modelliert. Im Folgenden werden diese detaillierten Pakete durch Tupel aus $\mathbb{R}^4$ beschrieben. Beispiele für detaillierte Pakete sind $p1^*$ = (1, 2, 5, 1), $p4^*$ = (1, 2, 5, 3) und $s3^*$ = (1, 4, 6 ,3).

Im Modell treten die Pakete durch einen Eingang in die Kommissionierung ein. Sie werden anschließend entsprechend ihrer Form in einen der beiden Puffer weitergegeben. Die Puffer erlauben wahlfreien Zugriff. Beim Eintreffen einer Bestellung, werden alle Pakete für den betreffenden Adressaten nacheinander in die eigentliche Kommissionieranlage gegeben. Diese versucht, die Pakete so dicht wie möglich anzuordnen. Abschließend wird die Sendung umverpackt und verlässt die Kommissionierung am Ausgang. Für jede Bestellung wird die Durchlaufzeit erfasst.

## 3.6 Inputs und Simulation der Kommissionierung

Auch für die Simulation der Kommissionierung wird ein entsprechender experimenteller Rahmen definiert. Hierzu gibt der Modellierer erneut Inputs an. Zum einen die Eingangspakete und zum anderen die Bestellungen.

Für die Bestellungen nutzt der Planer dieselbe Sequenz, wie bei der Simulation der gesamten Lieferkette (siehe 3.3). Um den Zufluss der Pakete zu modellieren, nutzt der Planer erneut Daten aus einem bestehenden Standort des Unternehmens. Hier werden neben dem Volumen auch die drei Dimensionen der Abmaße der Pakete erfasst. Das Ergebnis dieser Modellierung ist die zeitliche Abfolge der Eingangspakete.

Abbildung 3.5 zeigt die Abmaße dieser Pakete. Neben einer dreidimensionalen Darstellung (Perspektive) werden in der Abbildung auch eine Projektion auf Breite und Tiefe (Oben) sowie auf Höhe und Tiefe (Seite) abgebildet.

Insgesamt enthält diese Sequenz 1000 Pakete. Die Höhe der Pakete folgt einer Normalverteilung mit dem Mittelwert 2. Die Breite ist in zwei Cluster aufgeteilt. Dabei entfallen 70% der Pakete auf den linken Cluster und 30% der Pakete in den rechten Cluster. Innerhalb des Clusters ist die Höhe gleichverteilt. Zudem sind Höhe und Breite korreliert. Die Tiefe ist zwischen 0,2 und 5 gleichverteilt.

Abbildung 3.6 zeigt die Umsetzung des Modells mithilfe von Ptolemy II. Die Implementierung des Modells erfolgt ebenfalls mit dem Discrete-Time-Direktor. Entsprechend teilen sich das

Modell der Kommissionierung seine Ausführungssemantik mit dem Modell der gesamten Lieferkette (siehe Abbildung 3.1).

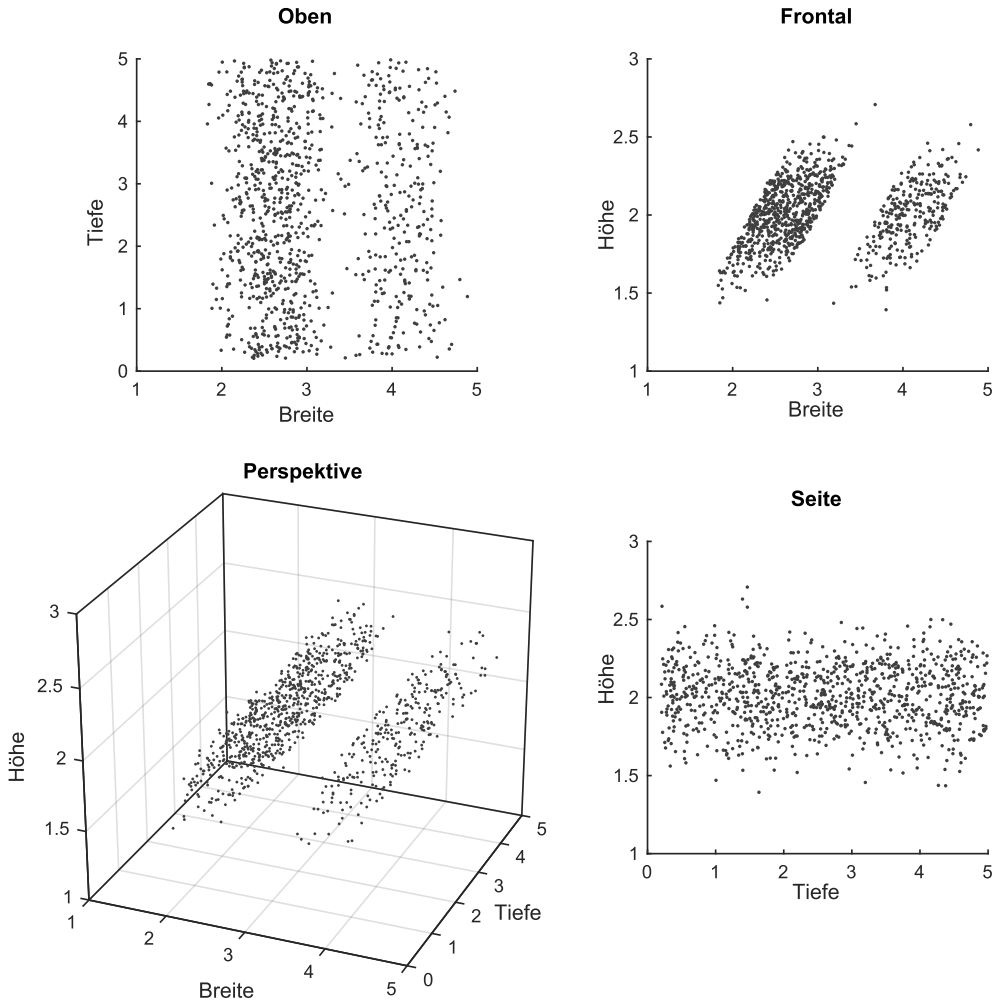



*Abbildung 3.5: Streu Diagramme des Inputsets.*

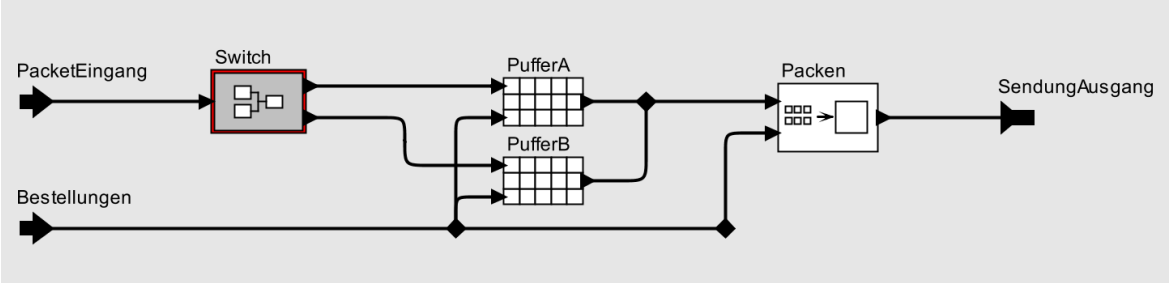



*Abbildung 3.6: Umsetzung des Modells der Kommissionierung in Ptolemy II.*

Das Modell der Kommissionierung wird mit der beschriebenen Inputsequenz simuliert. Die Durchlaufzeiten aller Sendungen werden ausgewertet. Es zeigt sich, dass 97% der Sendungen eine Durchlaufzeit von weniger als 3 Stunden haben. Die Anforderung A1.1 ist somit erfüllt.

## 3.7 Probleme im Beispiel

Der Planer hat den Feinentwurf für die Kommissionierung abgeschlossen und für sich genommen simulativ evaluiert. Er hält die Kommissionierung für den wichtigsten Standort der Lieferkette. Aus diesem Grund will er genauer untersuchen, wie der im Detail entworfene Standort im Zusammenspiel mit den anderen Standorten funktioniert. Offensichtlich ist es in diesem Szenario wichtig, diese Funktion vor der Integration der Lieferkette zu evaluieren, um auf eventuell auftretenden Problem noch reagieren zu können.

Konkret wirft der Planer dabei die folgenden Simulationsfragen auf:

***Erfüllt die Lieferkette die Anforderung A1, wenn die Kommissionierung sich aus den Teilen zusammensetzt, die im Feinentwurf definiert wurden?***

***Erfüllen die Teile der Kommissionierung die Anforderung A1.1, wenn sie mit den übrigen Standorten zur Lieferkette verbunden werden?***

Der Planer versucht, beiden Fragen erneut mit Kommissionierung* oder dem Modell der Lieferkette zu beantworten.

Kommissionierung* modelliert den Standort detailliert auf dem Abstraktionsniveau des Feinentwurfes. Das Modell beschreibt die Teile, aus denen sich die Kommissionierung zusammensetzt und die Interaktion zwischen diesen Teilen. Das ist erforderlich, schließlich zielen beide Fragen des Planers auf diese Teile ab. Somit weist das Modell den notwendigen Detailgrad auf, um die Frage zu beantworten. Allerdings hat es eine eingeschränkte Perspektive auf nur einen Standort. Die übrigen Standorte werden nur durch statische Inputsequenzen abgebildet. Dabei geht die Dynamik der Interaktion der drei Standorte untereinander verloren. Somit ist Kommissionierung* ungeeignet, um die beiden Fragen zu beantworten.

Das Modell der Lieferkette beschreibt alle Standorte der Lieferkette. Das ist wichtig zur Beantwortung der Fragen, da das Zusammenspiel aller Standorte für beide Fragen relevant ist. Das Modell weist die notwendige Ganzheitlichkeit auf. Allerdings berücksichtigt es nicht die Teile, aus denen sich die Kommissionierung zusammensetzt. Das wäre aber nötig, um Aussagen auf der Ebene des Feinentwurfes treffen zu können. Somit ist das Modell der Lieferkette ungeeignet, um die beiden Fragen zu beantworten.

***Beide Modelle sind für sich genommen zur Beantwortung der Simulationsfrage des Planers ungeeignet.***

Aus diesem Grund sucht der Planer einen Weg, die positiven Eigenschaften Ganzheitlichkeit und Detailgrad beider Modelle zu kombinieren.

Die beiden beschriebenen Modelle sind Beispiele dafür, wie es in Entwicklungsprozessessen zu Modellen unterschiedlicher Abstraktionsstufen kommt. Die beschriebenen Simulationsfragen geben eine beispielhafte Motivation für die Integration beider Ebenen. Die folgenden Abschnitte beschreiben prinzipielle Lösungsansätze, die es erlauben, Modelle unterschiedlicher Abstraktionsstufen zu verbinden.

## 3.8 Lösungsansatz: Offline-Kopplung

Ein Ansatz, um Modelle unterschiedlicher Abstraktionsstufen zu verbinden, liegt in einem manuellen Zyklus aus Analyse und Anpassungen.

Zunächst wird eines der beiden Modelle ausgeführt. Anschließend wird der Simulationsablauf des Modells analysiert. Die hierdurch gewonnenen Erkenntnisse werden dann genutzt, um das zweite Modell und seine Inputs entsprechend anzupassen. Nun erfolgt ein Simulationslauf des zweiten Modells, der ebenfalls analysiert wird. Schließlich werden die Erkenntnisse aus dieser zweiten Analyse in das erste Modell übernommen.

Während sie ausgeführt werden, bleiben die Modelle isoliert. Eine Kopplung erfolgt nur zwischen den Simulationsläufen. Deswegen werden derartige Lösungsansätze im Folgenden als Offline-Kopplung bezeichnet. Dieser Ansatz hat große Ähnlichkeit mit dem der wechselseitigen Kalibrierung wie er z.B. von Davis untersucht wird (Davis, 1995). Der Ansatz wird in Kapitel 5.1.1 näher beschrieben.

Abbildung 3.7 skizziert, wie diese Arten der Integration im Beispiel aussehen könnte. So kann der Planer nach einem Simulationslauf des Modells der Lieferkette die Taktung, mit der die Pakete das Lager verlassen, analysieren. Anschließend kann er die Inputsequenz von Kommissionierung* so anpassen, dass sie dieselbe Taktung aufweist. Nach der Simulation der Kommissionierung* kann er dann deren Outputs untersuchen, um die Verteilung des Aufschlags zu ermitteln. Diese Verteilung kann er anschließend im Modell der Lieferkette einsetzten.

Die Abbildung deutet zudem einen Zyklus an. Die Anpassungen der Inputs auf der linken Seite werden die Ausgaben des Modells Kommissionierung* verändern. Diese können wiederum die Ergebnisse der Analyse rechts ändern und so weiter. Aus diesem Grund kann es notwendig sein, diese Schleife mehrfach zu durchlaufen, bis die Analyseergebnisse stabil bleiben.

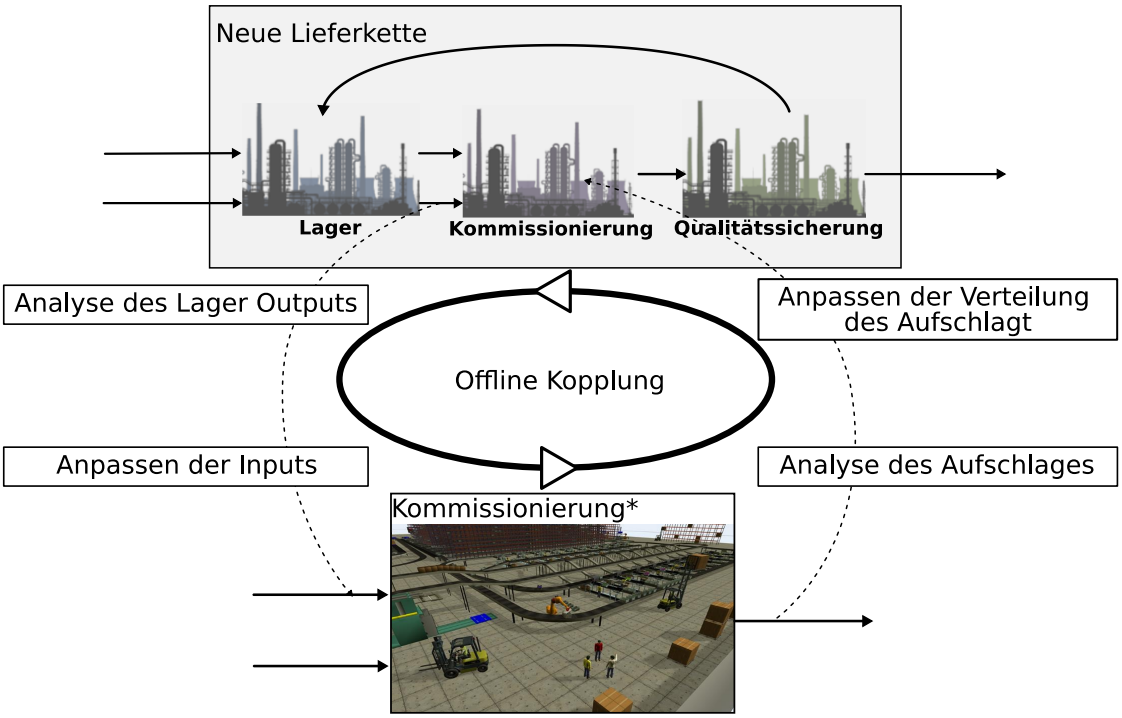



*Abbildung 3.7: Schematische Darstellung einer Offline-Kopplung des Beispiels*

### 3.8.1 Probleme

Die Offline-Kopplung ermöglicht es grundsätzlich, detaillierte Modelle im Kontext ihrer grob modellierten Umgebung zu untersuchen. Allerdigns weist die Offline-Kopplung Probleme auf, die im Folgenden diskutiert werden sollen.

So kann jede Änderung an einem der beiden Modelle zu Abweichungen des Simulationsablaufes führen. Da die Analyse auf dem letzten Simulationslauf beruht, muss sie ggf. ebenfalls neu durchgeführt werden. Letztlich kann auf diese Weise ein wiederholter Durchlauf der Offline-Kopplung notwendig werden. Wie bereits dargestellt muss diese unter Umständen dann sogar mehrfach durchlaufen werden, bis die Anpassungen stabil bleiben. ***Entwurfsänderungen können also einen immensen, zusätzlichen Aufwand für die Offline-Kopplung bedeuten.***

Ein weiteres Problem sind Rückkopplungen. Im Beispiel gibt es eine solche Rückkopplung zwischen den Ausgaben der Kommissionierung und ihren Eingaben über die Qualitätssicherung. Packt die Kommissionieranlage die Sendungen zu groß, durchlaufen sie die Kommissionierung mit hoher Wahrscheinlichkeit kurze Zeit später erneut. Rückkopplungen treten innerhalb eines Durchlaufes in jedem Zeitschritt auf. Die Offline-Kopplung wird aber nur zwischen zwei Durchläufen durchgeführt. ***Die Offline-Kopplung kann daher Rückkopplungen nur unzureichend erfassen.***

Es gibt keine Garantie, dass ein wiederholtes Durchlaufen des Zyklus zu einem Punkt führt, an dem die Analysen stabil bleiben. Es ist auch möglich, dass die Ergebnisse alternieren oder sogar divergieren. ***Es besteht die Gefahr, dass der Zyklus der Offline-Kopplung nicht terminiert.***

Für beiden Analyseschritte ist es notwendig zu wissen, was gesucht wird. Im Beispiel kennt der Planer die beiden Effekte bereits und nutzt die Simulation lediglich, um Sie zu quantifizieren. ***Deswegen ist der Ansatz nur begrenzt geeignet, um gänzlich unbekannte Effekte explorativ aufzudecken.***

Die Offline-Kopplung versucht, Eigenschaften, wie z.B. Verteilungen oder Taktungen, aus einem der Modelle auf das andere Modell zu übertragen. Daraus, wie dies geschieht, ergeben sich die genannten Probleme. Weil die Eigenschaften redundant sind, muss die Offline-Kopplung Änderungen an einem der Modelle nachpflegen. Weil die Kopplung immer nur zwischen Durchläufen geschieht, ist die Rückkopplung schlecht erfassbar. Weil die Eigenschaften, welche übertragen werden, explizit ausgewählt werden müssen, werden unbekannte Eigenschaften nicht übernommen.

Es stellt sich die Frage, ob es nicht möglich ist, ein Modell zu erzeugen, welches die Eigenschaften beider Modelle beinhaltet. Die Übertragung der Eigenschaften würde wegfallen.

## 3.9 Lösungsansatz: Vollständiges, detailliertes Modell

Eine verlockende Idee, um die Vorteile eines ganzheitlichen und eines detaillierten Modells zu verbinden, besteht darin, detaillierte Modelle aller Teilsysteme direkt miteinander zu verbinden. Es entsteht ein vollständiges, detailliertes Modell des Systems.

Dies kann technisch zum Beispiel durch ein Framework für Kosimulation wie FMI erreicht werden (siehe Abschnitt 2.2.1).

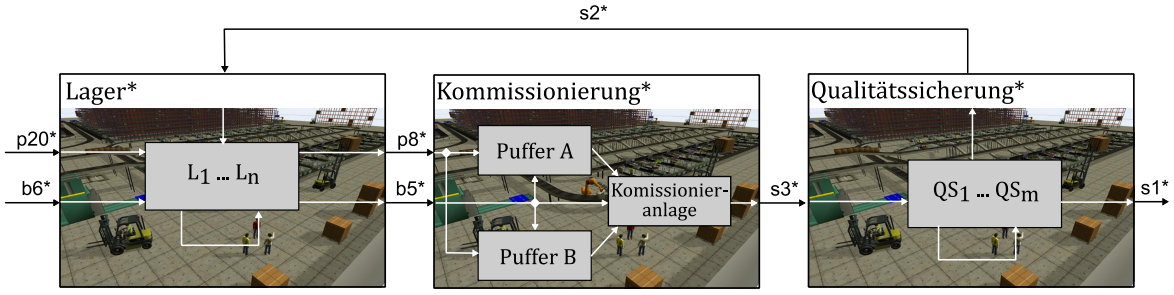



*Abbildung 3.8: Schematische Darstellung der gekoppelten Simulation aller Standorte*

Abbildung 3.8 skizziert ein derartiges Modell für das Beispiel. Das detaillierte Modell wird direkt mit den detaillierten Modellen Lager* und Qualitätssicherung* gekoppelt. Da wir an dieser Stelle nicht auf die konkrete Zusammensetzung dieser Standorte eingehen wollen,

werden die Teile, aus denen sie sich zusammensetzen, in der Abbildung lediglich als $L_1 \dots L_n$ bzw. $QS_1 \dots QS_m$ bezeichnet.

Das entstehende Modell ist ganzheitlich und detailliert. Im Beispiel ist es geeignet, um die Fragen aus Abschnitt 3.7 zu beantworten. Dabei können auch die Rückkopplung und unbekannte Effekte erfasst werden.

### 3.9.1 Probleme

Dennoch weist dieser Ansatz Probleme auf. Durch die Erstellung, Pflege und Simulation dieses vollständigen, detaillierten Modells ergeben sich erhebliche Modellierungs- und Rechenaufwände.

Im Beispiel müssen die detaillierten Modelle Lager* und Qualitätssicherung* modelliert und gepflegt werden. Es kann durchaus sein, dass bei der Erstellung des Feinentwurfs für das Lager kein detailliertes Modell des Lagers (Lager*) erforderlich ist. Somit müsste dieses Modell ausschließlich für die Kopplung erstellt und bei Änderungen des Feinentwurf aktualisiert werden. ***Es entsteht ein erheblicher zusätzlicher Modellierungsaufwand.***

Auch der Rechenbedarf, der für eine derartige vollständige Detailsimulation anfällt, ist für große Systeme problematisch. Aufgrund des detaillierten Modellierungsansatzes, der für diese Simulationen gewählt wurde, ist davon auszugehen, dass die notwendige Rechenleistung über dem Bedarf für die gröbere Simulation der Lieferkette liegt. ***Für komplexe Systeme mit einer Vielzahl von Teilsystemen kann dieser Ansatz einen zusätzlichen Rechen- und Modellierungsaufwand bedeuten.***

Beide Aufwände unterliegen in Entwicklungsprojekten einem Budget. Selbst wenn es also möglich sein sollte, eine vollständige, detaillierte Simulation durchzuführen, ist diese Lösung problematisch, weil die entsprechenden Ressourcen dann für andere Aufgaben fehlen.

Dabei wird unter Umständen Aufwand investiert, der zur Beantwortung der eigentlichen Simulationsfragen nicht erforderlich ist. Es ist möglich, dass nicht alle Teilsysteme für die aktuell betrachte Simulationsfrage relevant sind. Der Ansatz zwingt jedoch dazu, diese Teile dennoch zu modellieren und zu simulieren. Im Beispiel weiß der Planer etwa, dass die detaillierten, inneren Abläufe der Qualitätssicherung für die Kommissionierung bedeutungslos sind. Dennoch muss sie in jedem Simulationslauf mitbetrieben werden.

Es stellt sich die Frage, ob es möglich ist, die Modellierungs- und Rechenressourcen effizienter zu nutzen.

## 3.10 Lösungsansatz: Kopplung von Modellen mit Adaptern

Ein weiterer Ansatz, die unterschiedlichen Abstraktionsstufen miteinander zu verbinden, besteht darin, die Modelle der beiden Ebenen mithilfe von Adaptern zu koppeln.

Es gibt in der Literatur eine Reihe von Ansätzen, auf welche diese Beschreibung zutrifft. Sie werden in Abschnitt 5 vorgestellt und gegenübergestellt. Wesentliches Merkmal ist die Nutzung von Adaptern zur Verbindung der Ebenen.

Anders als bei dem in Abschnitt 3.9 beschriebene Ansatz, werden hierbei die Modellierungs- und Rechenressourcen effizient genutzt, da nur für die jeweils relevanten Teilsysteme ein detailliertes Modell betrieben wird.

Zur Erläuterung dieses Ansatzes, sowie der Probleme, die sich bei der Erstellung der Adapter ergeben, wird dieser Ansatz im weiteren Verlauf dieses Abschnittes auf das Lieferketten-Szenario angewendet.

Hierbei werden die Teilmodelle für das Lager und die Qualitätssicherung aus dem Modell der Lieferkette mit dem detaillierten Modell Kommissionierung* gekoppelt. Um dies zu erreichen, müssen jedoch beide Modelle entsprechend vorbereitet werden.

### 3.10.1 Modell der Lieferkette

Um das Teilmodell der Kommissionierung innerhalb des Modells der Lieferkette austauschen zu können, wird das Modell in Komponenten zerlegt. Diese Komponenten werden wiederum mit entsprechenden Schnittstellen versehen.

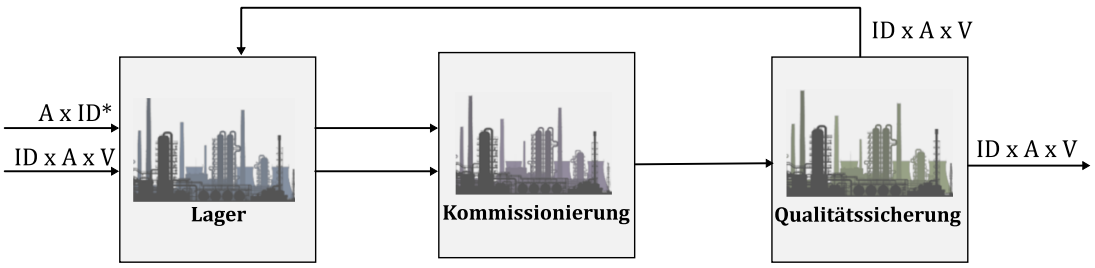



*Abbildung 3.9: Zerlegung des Modells der Lieferkette in Komponenten*

Abbildung 3.9 stellt die Zerlegung des Modells in die drei Komponenten Lager, Kommissionierung und Qualitätssicherung dar. Zur Darstellung der Schnittstellen werden erneut Pfeile genutzt. Sie werden mit dem Datentyp, der über die Schnittstelle übermittelt wird, beschriftet. Da es sich hierbei um eine technischere Darstellung als in Abbildung 3.1 handelt, werden die IDs explizit berücksichtigt.

Bei der Lieferkette werden grobe Pakete über diese Schnittstelle übertragen. Sie setzen sich, wie oben beschrieben, aus einer Identifikationsnummer, einem Adressaten und einem Volumen zusammen. Entsprechend werden $ID \subseteq \mathbb{I}$, $A \subseteq \mathbb{I}$ und $V \subseteq \mathbb{R}$ definiert.

Die Abbildung zeigt auch die Schnittstellen für die Bestellungen. Diese setzen sich erneut aus einem Adressaten A und einer endlichen Sequenz von Paket-Identifikationsnummern $ID^*$ zusammen.

In den frühen Phasen kann der Planer nun diese Komponenten koppeln, um den Grobentwurf zu evaluieren und zu untersuchen, ob die Anforderung A1 durch den Entwurf erfüllt wird. Hierbei verbindet er die Komponenten wie in der Abbildung 3.9 dargestellt. Die Inputs werden an den freien Schnittstellen eingespeist. Anschließend kann der Planer diese gekoppelten Modelle genauso nutzen wie zuvor das Modell der Lieferkette.

### 3.10.2 Kommissionierung*

Auch das Modell der Kommissionierung interpretiert der Planer als Komponente mit Schnittstellen. Abbildung 3.10 zeigt Kommissionierung* als Komponente. Die Typen ihrer Schnittstellen entsprechen den bekannten Bestellungen und den detaillierten Paketen. Bei den detaillierten Paketen wird nicht das Volumen, sondern deren Dimensionen modelliert. Diese werden wie folgt definiert:

$$H \times B \times T \subseteq \mathbb{R}^3$$

Hierbei stehen H, B und T für Höhe, Breite und Tiefe des Paketes.

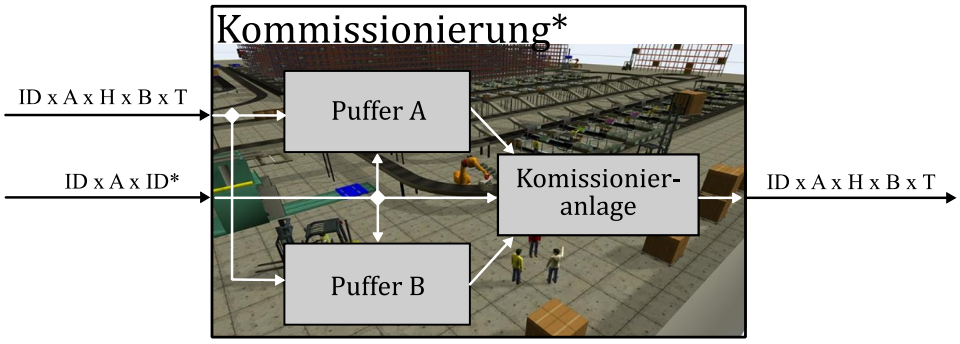



*Abbildung 3.10: Das Modell Kommissionierung* als Komponenten.*

Um den Feinentwurf der Kommissionierung isoliert zu evaluieren, kann der Planer weiterhin nur dieses Modell simulieren.

### 3.10.3 Kopplung im Beispiel

Nachdem der Planer den Feinentwurf evaluiert hat, konnte er bestätigen, dass der Entwurf isoliert die Anforderung A1.1 erfüllt. Nun will er das Zusammenspiel der Modelle untersuchen. Hierzu koppelt er die Lager- und Qualitätssicherungskomponenten mit Kommissionierung*.

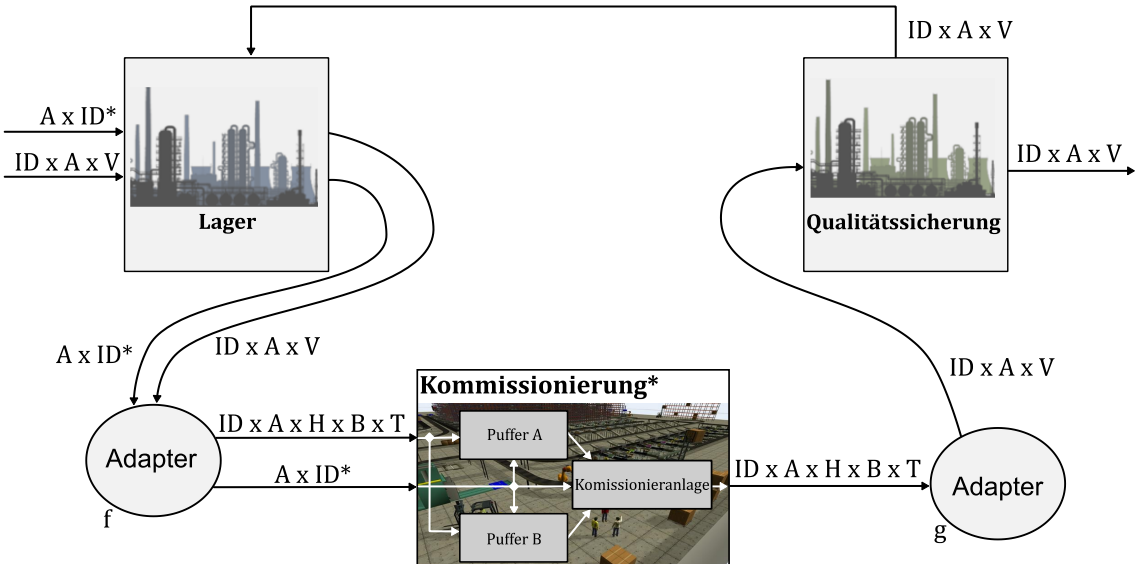



*Abbildung 3.11: Kopplung des Beispiels auf Basis von Adaptern.*

Abbildung 3.11 zeigt wie diese Kopplung erreicht werden kann. Der Planer hat die grobe Komponente Kommissionierung deaktiviert. An ihre Stelle tritt die detaillierte Komponente Kommissionierung*.

Die beiden Komponenten Lager und Qualitätssicherung verfügen jedoch nicht über Schnittstellen desselben Typs wie Kommissionierung*. Aus diesem Grund ist der Planer gezwungen, Adapter zu programmieren. Diese haben die Aufgabe, beide Nachrichtentypen ineinander umzuwandeln.

### 3.10.4 Adapter im Beispiel

Der Planer benötigt zwei Adapter, welche die Datentypen des Modells der Lieferkette und des detaillierten Modells der Kommissionierung ineinander konvertieren. Bei genauerer Betrachtung stellt man fest, dass die Bestellungen, sowie die Identifikationsnummer und der Adressat der Pakete, unkritisch sind. Eine Konvertierung ist nicht erforderlich.

Die eigentliche Aufgabe der Adapter liegt darin, zwischen dem Volumen und den Dimensionen der Pakete abzubilden. Der Planer sucht im Wesentlichen zwei Funktionen. Wir wollen diese im Folgenden als f und g bezeichen. Dabei sei

$$f: V \rightarrow H \times B \times T$$

die Abbildung der groben Pakete des Modells der Lieferkette auf die detaillierten Pakete des Modells Kommissionierung*. Zudem sei

$$g: H \times B \times T \to V$$

die Abbildung der detaillierten Pakete des Modells Kommissionierung* auf die groben Pakete des Modells der Lieferkette. Beide werden in Abbildung 3.11 neben dem Adapter, in dem sie genutzt werden, dargestellt.

Der Planer beginnt mit der Beschreibung von g. Das Volumen eines Paketes kann durch das Produkt von Höhe, Breite und Tiefe bestimmt werden. Entsprechend einfach ist es, g explizit anzugeben.

$$\mathrm{g}(\mathrm{h}, \mathrm{b}, \mathrm{t}) = \mathrm{h} * \mathrm{b} * \mathrm{t}$$

Nun muss der Planer noch ein geeignetes f finden. Er weiß, dass ein Paket, welches die Kommissionierung betritt und ohne jegliche Umverpackung wieder verlässt, sein Volumen nicht ändern darf. Die Hintereinanderausführung von zuerst f und anschließend g muss also wieder zu demselben groben Paket führen.

Die Verkettung von g und f muss die Identität auf den Volumen V ergeben.

$$\mathrm{g} \circ \mathrm{f} = \mathrm{id}_{\mathrm{V}}$$

Der Planer wählt eine Funktion, welche diese Eigenschaften erfüllt:

$$\mathrm{f}(\mathrm{v}) = \begin{pmatrix} \mathrm{v}^{\frac{1}{3}} \\ \mathrm{v}^{\frac{1}{3}} \\ \mathrm{v}^{\frac{1}{3}} \end{pmatrix}$$

Wie nun leicht ersichtlich, ergibt die Verkettung von g und f die Identität auf V.

$$\mathrm{g}\big(\mathrm{f}(\mathrm{v})\big) = \mathrm{g}\left(\begin{pmatrix} \mathrm{v}^{\frac{1}{3}} \\ \mathrm{v}^{\frac{1}{3}} \\ \mathrm{v}^{\frac{1}{3}} \end{pmatrix}\right) = \mathrm{v}^{\frac{1}{3}} * \mathrm{v}^{\frac{1}{3}} * \mathrm{v}^{\frac{1}{3}} = \mathrm{v}^{\left(\frac{1}{3}+\frac{1}{3}+\frac{1}{3}\right)} = \mathrm{v}$$

### 3.10.5 Probleme

Der Ansatz verbindet die Vorteile der vorherigen Ansätze, vermeidet dabei jedoch ihre Nachteile. Die Adapter müssen nur einmal angegeben werden, Rückkopplungen werden

korrekt abgebildet und prinzipiell können auch unbekannte Effekte gefunden werden (siehe Probleme der Offline-Kopplung 3.8.1).

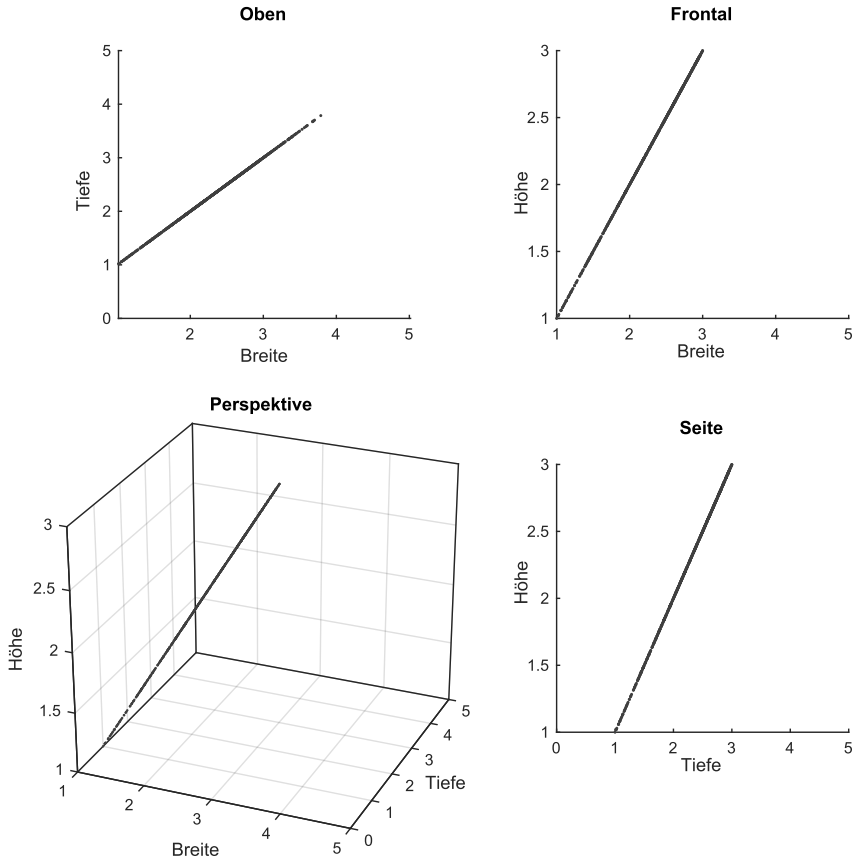



*Abbildung 3.12: Ausgabe der Funktion f.*

Zudem wird bei einer Simulation nur das detaillierte Modell der relevanten Teilsysteme benötigt. Auf detaillierte Modelle der übrigen Teilsysteme kann verzichtet werden (siehe Probleme bei Kopplung aller detaillierten Modelle 3.9.1).

Doch auch dieser Ansatz hat im Beispiel ein grundlegendes Problem. Abbildung 3.12 stellt die Abmessungen aller detaillierter Pakete dar, die von f während eines Simulationslaufes erzeugt werden. Die beschriebene, naive Funktion f erzeugt immer kubische Pakete, was offensichtlich nicht korrekt ist. Sie unterscheiden sich stark von den Inputs, die der Planer für die Detailsimulation der Kommissionierung modelliert hat (siehe Abbildung 3.5).

In der Frage, wie eine Funktion $f$ aussehen muss, liegt das zentrale Problem des Ansatzes. Aufgrund der Abstraktionsbeziehung zwischen den Schnittstellen der unterschiedlichen Ebenen, hat $f$ mehrere Konkreta zur Auswahl, auf die es ein Abstraktum abbilden kann. Dies realisiert sich auch im Beispiel.

Betrachten wir hierzu ein grobes Paket mit dem Volumen 8.

$$f(8) = \begin{pmatrix} 8^{\frac{1}{3}} \\ 8^{\frac{1}{3}} \\ 8^{\frac{1}{3}} \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

Wie bereits dargelegt führt die Verkettung mit g wieder zu unserem Ausgangsvolumen.

$$g\left(\begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}\right) = 2 * 2 * 2 = 8$$

Allerdings gibt es noch weitere Kombinationen, die zu dem Volumen 8 führen können.

$$1 * 2 * 4 = 8, \qquad 8 * 1 * 1 = 8$$

Tatsächlich gibt es unendlich viele solcher Kombinationen. Für jede Kombination aus *Tiefe* und *Breite* kann eine *Höhe* gefunden werden so das $\text{Höhe} * \text{Breite} * \text{Tiefe} = 8$. Durch Umformen erhält man leicht eine entsprechende Berechnungsvorschrift.

$$\text{Höhe} = \frac{8}{\text{Tiefe} * \text{Breite}}$$

Abbildung 3.13 zeigt eine Oberfläche mit allen Kombinationen der drei Dimensionen, welche das Volumen 8 ergeben. Jeder Punkt auf der dargestellten Oberfläche ist eine gültige Ausgabe für f(8) so das $g(f(8)) = 8$.

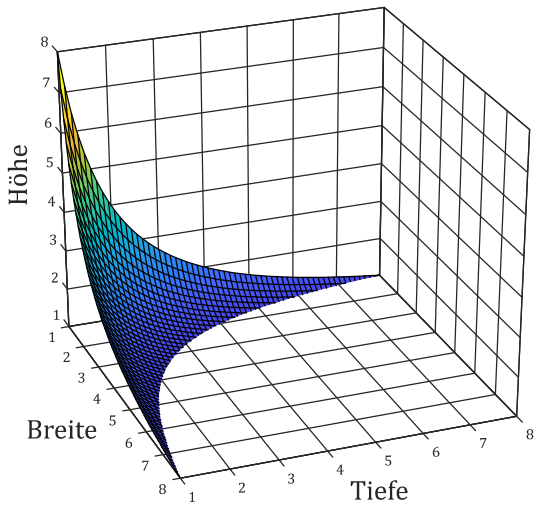



*Abbildung 3.13:Darstellung der Dimensionen aller Pakete mit Volumen 8.*

***Wie soll ein Adapter zwischen allen diesen potenziellen Kandidaten den „richtigen“ auswählen?***

Die Menge der „zulässigen“ detaillierten Pakete ist - wie beschreiben - aufgrund der realwertigen Dimensionen der Pakete unendlich groß. Selbst wenn man diese Menge z.B. durch Runden auf eine endliche Teilmenge beschränken würde, wäre das Problem nicht gelöst. Im Beispiel muss die Funktion ein grobes Paket in genau ein detailliertes Paket umwandeln. Also muss die Funktion ein einziges Paket aus den zulässigen Paketen auswählen.

***Was sind Kriterien für diese „richtige“ Ausgabe?***

Bis jetzt wurde im Beispiel nur ein einziges Kriterium für die Ausgabe von $f$ definiert. Die Verkettung von g mit f muss die Identität bilden.

$$g \circ f = id_V$$

Das gilt aber für alle Punkte in Abbildung 3.13 gleichermaßen. Es müssen also noch weiter Kriterien für die „richtige“ Ausgabe gefunden werden.

***Wie kann der Planer einen solchen Adapter angeben, ohne übermäßigen Modellierungsaufwand zu investieren?***

Wie wir gerade gesehen haben, ist der naive Ansatz, eine determinierte Funktion für den Adapter anzugeben, gescheitert. Es muss ein Weg gesucht werden, auf dem das Wissen des Planers über die Inputs der Kommissionierung in den Adapter einfließen können.

Hierbei besteht aber die Gefahr, dass dies wiederum zu einem problematischen Modellierungs- und Rechenaufwand führt. Der Aufwand zur Beschreibung und Anwendung des Adapters muss deutlich kleiner sein als die detaillierte Simulation aller Standorte (in Abschnitt 3.9). Ansonsten verliert die Adapter-Lösung ihre Vorteile gegenüber einer vollständigen Detailsimulation.

Die Nutzung von Adaptern zur Überbrückung der Abstraktionsebenen ist der Kerngedanke des Ansatzes und daher ist die Frage, wie diese Adapter beschaffen sind, wann sie „gut“ sind und wie sie erstellt werden, von zentraler Bedeutung.

# 4 Analyse von Anforderungen an eine Multi-Level-Simulation

Das Problem aus dem Beispiel der Lieferkette steht stellvertretend für ein allgemeines Problem einer Kopplung von Modellen verschiedener Abstraktionsstufen. Naiv formuliert, geht bei dem Übergang in den abstrakteren Zustandsraum des Grobmodells Information verloren, die bei der Abbildung in den konkreteren Zustandsraum des Detailmodells wieder „aufgefüllt“ werden muss. Das in 3.10 beschriebene Konzept, Adapter durch eine Funktion zu definieren, umgeht dieses Problem, indem es eine eindeutige Antwort für dieses „Auffüllen“ erzwingt. Das Ergebnis ist jedoch, dass nur ein winziger Teil der möglichen Detailstände erreicht wird. Im Beispiel nur kubische Pakete.

In diesem Abschnitt wird die Notwendigkeit für ein „Auffüllen“ auf die Beschaffenheit der Abstraktionsbeziehung zwischen den Variablen von Grob- und Detailsimulation zurückgeführt.

Hierzu werden zunächst in Abschnitt 4.1 die Elemente einer Abstraktionsebene übergreifenden Simulation herausgearbeitet. Anschließend werden in Abschnitt 4.2 die zwischen diesen Elementen herrschenden Abstraktionsbeziehungen charakterisiert.

Im Anschluss werden zwei zentrale Anforderungen an eine solche Multi-Level-Simulation abgeleitet. In Abschnitt 4.3.1 wird im Rahmen der erarbeiteten Struktur und unter Berücksichtigung der Abstraktionsbeziehungen eine Konsistenzanforderung aufgestellt. Abschnitt 4.3.2 zeigt schließlich, wie die Abstraktionsbeziehung zu dem beschriebenen Problem bei der Definition einer Zustandsabbildung „nach unten“ führt und leitet eine Anforderung an die Verteilung der Konkreta, die durch den Adapter während einer Multi-Level-Simulation erzeugt werden, ab.

## 4.1 Elemente einer Multi-Level-Simulation

Den Rahmen für die untersuchte Klasse von Simulationsstudien stellen zwei Beschreibungsebenen eines Systems. Eine Grobentwurf beschreibt die Teilsysteme des Systems und bringt sie in Zusammenhang. Ein Feinentwurf beschreibt die Elemente, aus denen sich wiederum die Teilsysteme zusammensetzen. Im Beispiel ist ein solches Teilsystem die Kommissionierung. Ein Beispiel für ein Element, aus dem das Teilsystem besteht, ist der Puffer A.

Gemäß dem V-Modell des Entwicklungsprozesses (VDI, 2003, S. 16), werden die Feinentwürfe nicht mehr für das gesamte System, sondern je ein Feinentwurf für die einzelnen Teilsysteme ausgearbeitet.

Beiden Entwürfen werden im Sinne der Durchgängigkeit (VDI, 2003, S. 26) als Modell abgebildet. In Kapitel 3 wurden diese Modelle für das Lieferkettenbeispiel detailliert beschrieben. Aufgrund des jeweiligen Betrachtungsgegenstandes wird das Modell des Grobentwurfes im Folgenden als ***Systemmodell*** und das Modell eines Feinentwurfes als ***Teilsystemmodell*** bezeichnet. Das Teilsystemmodell beschreibt einen kleinen Systemausschnitt und erlaubt so eine detailliertere Modellierung als das weiter gefasste Systemmodell.

Abbildung 4.1 stellt diese Einteilung für die beiden Modelle des Lieferkettenbeispiels sowie die zugehörigen Entwürfe gegenüber.

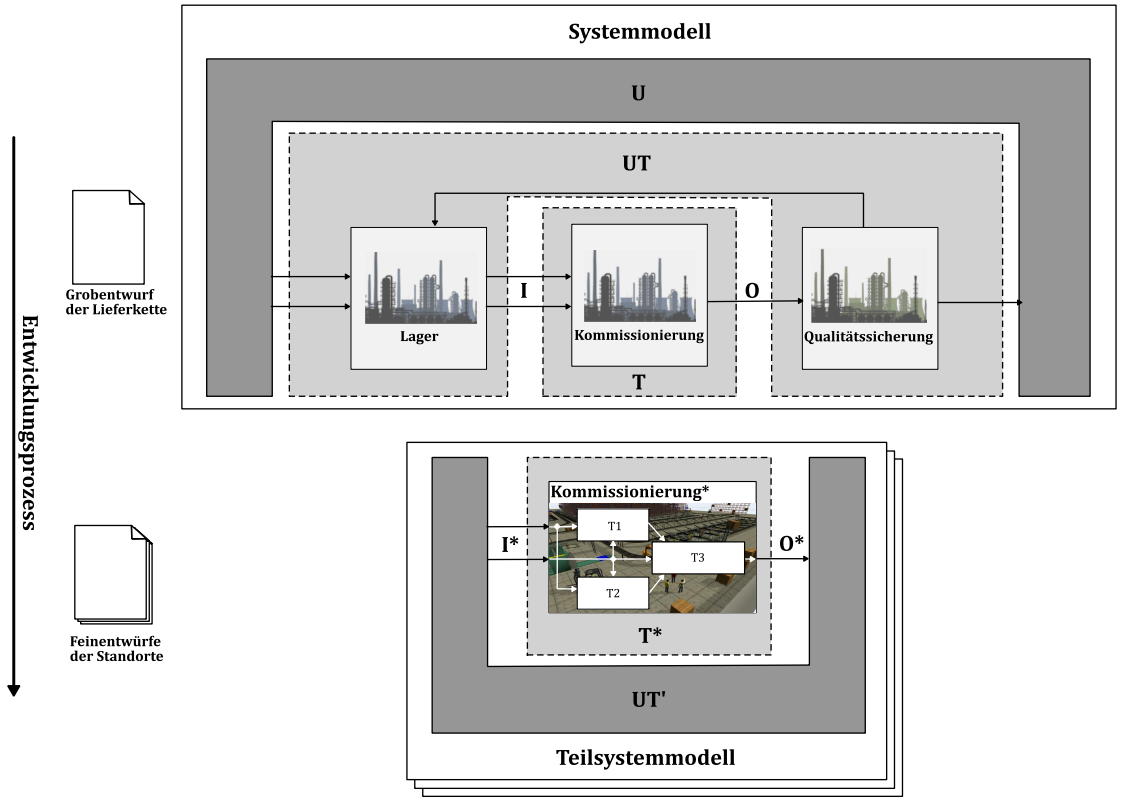



*Abbildung 4.1: Systemmodell und Teilsystemmodell im Lieferkettenbeispiel*

**Systemmodell.** Betrachten wir zunächst das Systemmodell. Im Systemmodell werden die Teilsysteme sowie deren Zusammenhang miteinander modelliert.

Wählen wir ohne Beschränkung der Allgemeinheit eines der Teilsysteme für unsere Betrachtung aus. Wie bereits beschrieben, ist die Kommissionierung ein Beispiel für ein solches Teilsystem. Im Folgenden wird der Bereich des Systemmodells, welcher unser Teilsystem beschreibt, als $T$ bezeichnet. In Abbildung 4.1 wird der betreffende Modellteil des Beispiels entsprechend eingerahmt und beschriftet.

Das Gesamtmodell besteht neben $T$ noch aus Modellen aller weiteren Teilsysteme. Wir können ebenfalls ohne Beschränkung der Allgemeinheit alle diese weiteren Teilsysteme als Umgebung des Teiles $UT$ zusammenfassen. Im Beispiel können das Lager und die Qualitätssicherung in $UT$ zusammengefasst werden. $T$ und $UT$ interagieren miteinander über die Variablen $I$ und $O$. $I$ umfasst dabei alle Inputs von $UT$ an $T$. $O$ umfasst alle Outputs von $T$ an UT. Gemeinsam stellen $T$ und $UT$ ein Modell des Systems dar.

Auch das System als Ganzes ist in eine Umgebung eingebettet. Ein Modell dieser Umgebung ist über Schnittstellen mit dem Systemmodell verbunden. Im Beispiel ist die Umgebung durch die Abfolge der Inputs für den Wareneingang und die Bestellungen beschrieben. Die Inputsequenz ist nichts Anderes als ein statisches Modell der Umgebung. In Abbildung 4.1 wird die Umgebung des Systems mit U bezeichnet.

Die Umgebung $U$ liefert den experimentellen Rahmen der Simulationsstudie.

**Teilsystemmodell.** Betrachten wir nun das zu dem gewählten Teilsystem gehörige Teilsystemmodell. Auch das Teilsystem setzt sich laut Feinentwurf aus bestimmten Elementen zusammen. Das Teilsystemmodell beschreibt die Elemente des Teilsystems und wie diese interagieren. Die Menge dieser Elemente des Teilsystems wird im Folgenden als $T^*$ bezeichnet. Auch das Teilsystemmodell ist mit einem Modell seiner Umgebung verbunden. Nennen wir diese Umgebung UT‘. Erneut fassen wir unter $I^*$ beziehungsweise $O^*$ alle Variablen, über die $T^*$ und $UT'$ interagieren, zusammen. Das Teilsystemmodell setzt sich somit aus $T^*$ und UT‘ zusammen.

UT‘ stellt den experimentellen Rahmen der Simulationsstudie da.

Das detaillierte Modell $Kommissionierung^*$ ist ein Beispiel für ein solches Teilsystemmodell. Die beiden Puffer und die Kommissionieranlage sind Beispiele für diese Elemente. $UT'$ wird im Beispiel durch eine Inputsequenz beschrieben. Die Eingänge für Bestellungen und Pakete des Modells Kommissionierung* können unter $I^*$ zusammengefasst werden. Analog besteht $O^*$ lediglich aus dem Ausgang der Sendungen.

## 4.2 Arten der Abstraktion

Der Begriff der Abstraktion wird häufig in verschiedensten Bedeutungen verwendet. Tatsächlich werden unterschiedliche Zusammenhänge als Abstraktion beschrieben. Im Beispiel haben wir das intuitive Begriffsverständnis verwendet, nach dem der Grobentwurf und das Systemmodell eine abstraktere Beschreibung darstellen als die Feinentwürfe und die Teilsystemmodelle. Die nun vorliegende Strukturierung der Elemente einer Multi-Level-Simulation erlaubt es uns, den Begriff der Abstraktion differenziert zu betrachten.

Diese Differenzierung erfolgt auf Basis der Arbeiten von Frantz (Frantz, 1995) und Zeigler (Zeigler, 2019, Kap. 16) zum Abstraktionsbegriff in Simulationsmodellen. Im folgenden Abschnitt werden aus den weitergefassten Taxonomien dieser Autoren die drei für die in 4.1 identifizierte Elemente relevanten Facetten des Abstraktionsbegriffes ableiten. Wir werden diese Facetten als Approximation, Aggregation und Variablenabstraktion bezeichnen.

Betrachten wir zunächst in Abbildung 4.2 erneut die Elemente einer Multi-Level-Simulation die in Abschnitt 4.1 analysiert wurden. In der Abbildung treten drei Elemente zweifach auf. Die gestrichelten, farbigen Linien verbinden jeweils die Stellen, an denen die Elemente auftreten.

Die Umgebung des Teilsystems wird sowohl durch UT im Systemmodell als auch durch UT‘ im Teilsystemmodell beschrieben (grüne Linie). Das ausgewählte Teilsystem wird durch T und T* modelliert (rote Linie). Zudem beschreiben I und $O$ im Systemmodell sowie $I^*$ und $O^*$ im Teilsystemmodell jeweils die Ein- und Ausgänge zwischen Teilsystem und Umgebung des Teilsystems (blaue Linien). Aufgrund der Gemeinsamkeiten bezüglich des Abstraktionsbegriffes, werden Ein- und Ausgänge im Folgenden gemeinsam diskutiert.

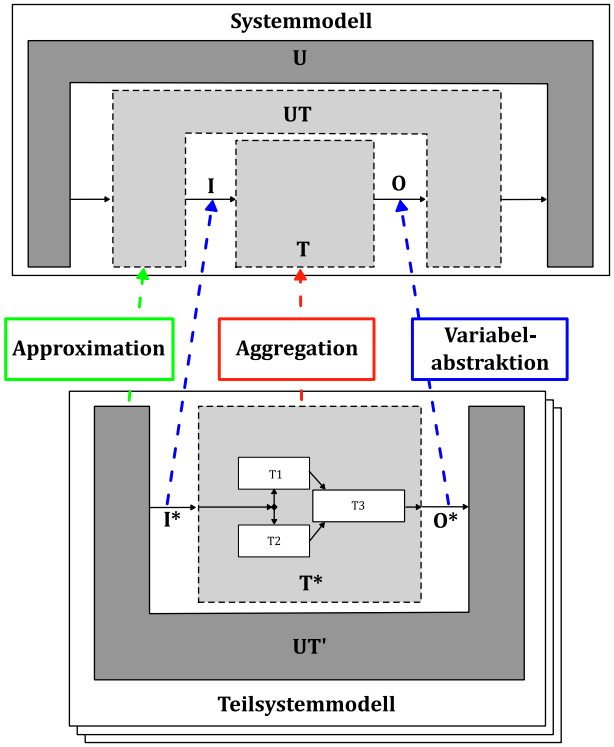



*Abbildung 4.2: Abstraktion im Beispiel*

Somit können drei Paare identifiziert werden, die Kandidaten für das Auftreten von Abstraktion sind. Wie wir später sehen werden, handelt es sich bei jedem der Paare um eine andere Form der Abstraktion. Im Folgenden werden diese Paare diskutiert und eine Definition des jeweiligen Abstraktionsbegriffes erarbeitet.

### 4.2.1 Approximation

Die Umgebung des Teilsystems wird sowohl durch UT als auch durch UT‘ beschrieben.

$UT'$ stellt eine Umwandlung von $UT$ in eine einfachere Beschreibungsform dar. Hierdurch wird insbesondere Rechenzeit eingespart.

Auch im Lieferkettenbeispiel stellt $UT'$ eine derartige Vereinfachung von $UT$ da. In UT wird die Umgebung durch die Interaktion zwischen den Standorten Lager und Qualitätssicherung modelliert. In UT′ werden diese komplexen Zusammenhänge in Form einer statischen Sequenz von Inputs abgebildet. Die Inputsequenz des Beispiels kann als Modell ohne Eingabe aufgefasst werden, welches bei jedem Aufruf das nächste Element der Sequenz ausgibt. Intuitiv würden wir diese Form der Modellierung als abstrakter gegenüber der Beschreibung in UT begreifen.

Tatsächlich lässt sich in der Taxonomie von Frantz (Frantz, 1995) ein zu dem Zusammenhang zwischen $UT$ und $UT'$ passender Eintrag finden. Unter einer *Modification of Model Form* fasst Frantz Techniken zusammen, die dadurch charakterisiert werden, dass Sie die Input-Output-Transformation des Modells vereinfachen. Dabei wird das Modell als Blackbox betrachten. Die Funktion, welche die Inputs der Blackbox auf ihre Outputs abbildet, wird approximiert. Auch Zeigler beschreibt mit *linearization* und *formalism transformation* (Zeigler, 2019, S. 409) vergleichbare Konzepte.

Diese Umwandlung eines Modells in eine vereinfachte Beschreibungsform wird im weiteren Verlauf als ***Approximation*** bezeichnet. Das Ziel der Approximation ist es, Rechenzeit zu sparen.

UT′*(des Teilsystemmodells)* ***ist eine Approximation von*** UT *(des Systemmodells).*

Eine einfache Art der Approximation ist die Verwendung einer Lookup-Table, wie sie im Lieferkettenbeispiel genutzt wurde. Das Modell kann auch mit verschiedenen anderen Mechanismen vereinfacht werden. So können zwischen den Stützpunkten einer Lookup-Table Werte mit einem beliebigen Interpolationsverfahren ermittelt werden. Auch der Einsatz von Verfahren des maschinellen Lernens für die Approximation der Modellfunktion ist möglich (z.B. (Engel, 2019)).

Eine mögliche Betrachtungsweise einer Multi-Level-Simulation ist, dass hierbei die approximierte Umgebung $UT'$ durch $UT$ ersetzt wird.

### 4.2.2 Aggregation

Das Teilsystem wird sowohl durch $T$ als auch durch $T^*$ beschreiben. Im Systemmodell wird das Teilsystem als eine einzelne Entität $T$ modelliert. Dem gegenüber wird das Teilsystem

innerhalb des Teilsystemmodells durch eine Menge von Elementen $T_i$ modelliert. Das Verhalten des Teilsystems ergibt sich aus dem Zusammenspiel dieser Elemente.

Im weiteren Verlauf dieser Arbeit wird diese Art der Abstraktion als ***Aggregation*** bezeichnet.

***Zwischen*** T ***und*** $T^*$ ***besteht eine Aggregation.***

Im Beispiel werden die Teile *Puffer A*, *Puffer B*, und *Kommissionieranlage* im Teilsystemmodel zur *Kommissionierung* des Systemmodells aggregiert.

Das monolithische Teilsystem aus $T$ wird in $T^*$ in seine Elemente zerlegt. Diese Elemente entsprechen dabei der Strukturierung, die der Feinentwurf des Teilsystems vorsieht. Intuitiv würden wir daher $T$ als abstrakter einstufen als das $T^*$. Auch wenn $T$ vor $T^*$ modelliert wurde, können wir $T$ als eine Abstraktion von $T^*$ verstehen. Es besteht eine hierarchische Beziehung zwischen dem Teilsystem aus $T$ und den Elementen T1, T2 und T3.

In der Arbeit von Zeigler wird diese Abstraktionsform ebenfalls als Aggregation bezeichnet und wie folgt beschrieben:

> *combining groups of components into a single component which represents their combined behavior when interacting with other groups* (Zeigler, 2019, S. 409)

Auch in der Taxonomie von Frantz (Frantz, 1995) ist diese Art der Abstraktion zu finden. Unter der Hauptkategorie *Modifikation of Behavior* beschreibt er die *Entity Aggregation*. Hierbei werden Modellelemente eines „niedrigeren Levels“ (z.B. T1 T2 T3) zu Modellelementen eines „höheren Levels“ (z.B. $T$) aggregiert.

### 4.2.3 Variablenabstraktion

Die Ein- und Ausgänge zwischen dem Teilsystem und seiner Umgebung werden sowohl durch $I$ und $O$ im Systemmodell als auch durch $I^*$ und $O^*$ im Teilsystemmodell beschrieben. Die zentrale Eigenschaft der Abstraktionsbeziehung zwischen beiden Ebenen liegt darin, dass es für eine Belegung der Variablen in $I$ und $O$ mehrere äquivalente Belegungen der Variablen in $I^*$ bzw. $O^*$ gibt.

Intuitiv würden wir daher $I$ als Abstraktion von $I^*$ verstehen. Tatsächlich entspricht die Beziehung zwischen Belegungen der Variablen in $I$ und $I^*$ dem mathematische Abstraktionsbegriff. Belegungen von $I^*$ werden in diesem Zusammenhang als Konkreta und Belegungen von $I$ als Abstrakta bezeichnet.

Die Abstrakta werden beim mathematischen Abstraktionsbegriff mit Äquivalenzklassen identifiziert. Ausgehend von einer Menge von Konkreta wird ein Äquivalenzrelation (eine reflexive, transitive, symmetrische Relation) definiert. Diese partitioniert die Menge der Konkreta in Äquivalenzklassen. Jede dieser Klassen ist genau eines der möglichen Abstrakta.

Übertragen auf unser Beispiel heißt dies, dass die konkreten Pakete genau den Konkreta entsprechen. Als äquivalent werden alle Pakete mit demselben Volumen verstehen. In der Äquivalenzklasse eines Paketes sind dann genau alle jene Pakete mit demselben Volumen. Abbildung 3.13 zeigt die Äquivalenzklasse des Paketes $\begin{pmatrix}2\\2\\2\end{pmatrix}$, also alle Pakete mit Volumen 8.

Im Rahmen dieser Arbeit ist es jedoch zweckmäßig, Abstraktion nicht direkt über eine Relation zwischen den Konkreta zu definieren. Stattdessen definieren wir die Variablenabstraktion[1] zunächst als eine surjektive Abbildung zwischen Konkreta und Abstrakta. Anschließend kann die Äquivalenzrelation auf Basis dieser Abbildung definiert werden.

***Eine Variablenabstraktion ist eine Surjektion*** $\alpha: K \rightarrow A$***, die von einer Menge von Konkreta K auf eine Menge von Abstrakta A abbildet.***

Die Surjektivität kann wie folgt formuliert werden.

$$\forall a \in A\ \exists k \in K: \alpha(k) = a$$

Dies stellt sicher, dass es möglich ist, eine Äquivalenzrelation zu definieren. Ein Abstraktum ohne Konkreta entspräche einer leeren Äquivalenzklasse. Das ist nicht möglich.

Gäbe es ein Konkretum *k* mit der Äquivalenzklasse $[k]_{\sim} = \emptyset$, so wäre $k \notin \emptyset = [k]_{\sim}$ und somit $k \notin [k]_{\sim}$. Aus der Definition einer Äquivalenzklasse ($[k]_{\sim} := \{x \in K \mid x \sim k\}$) ergibt sich wiederum, dass k nicht mit sich selbst äquivalent ist. Somit wäre ~ nicht reflexiv und keine Äquivalenzrelation. Nun kann die zu $\alpha$ gehörige Äquivalenzrelation wie folgt definiert werden:

$$\sim_{\alpha} \subseteq K \times K$$

$$k_1 \sim_{\alpha} k_2 :\Leftrightarrow \alpha(k_1) = \alpha(k_2)\ mit\ k_1, k_2 \in \widehat{K}$$

[1] Die Beziehung besteht zwischen Belegungen der Variablen. Deshalb ist Variablenbelegungsabstraktion der korrektere Begriff. Aus Gründen der Lesbarkeit wird im weiteren Verlauf der Arbeit jedoch der kürzere Begriff Variablenabstraktion verwendet.

Es ist leicht ersichtlich das diese Relation die notwendigen Eigenschaften Reflexivität, Symmetrie und Transitivität aufweist. Approximation und Aggregation beziehen sich beide auf die Beschreibungsform und die Struktur der Modelle. Sie sind insbesondere während der Modellierung relevant. Demgegenüber besteht eine Variablenabstraktion zwischen Belegungen von Variablen während der Simulation von Modellen. Der Variablenabstraktion kommt im weiteren Verlauf der Arbeit eine zentrale Rolle bei der Ableitung der Anforderungen für eine Multi-Level-Simulation zu.

## 4.3 Anforderungen an eine Multi-Level-Simulation

Versucht man, Modelle unterschiedlicher Abstraktionsebenen im Rahmen einer Multi-Level-Simulation zu integrieren, wird eine vorliegende Variablenabstraktion zur zentralen Herausforderung. Im Lieferkettenbeispiel tritt sie zwischen abstrakten und detaillierten Paketen auf. Wie bereits dargestellt, gibt es hier zu jedem abstrakten Paket $a$ nicht ein einziges, konkretes Gegenstück $k$, sondern eine ganze Äquivalenzklasse $[\mathrm{k}]_{\sim}$.

Die Verteilung der Konkreta innerhalb einer Äquivalenzklasse ist dabei von entscheidender Bedeutung für die Validität der Multi-Level-Simulation. Das Beispiel hat gezeigt, dass der Ansatz, die Äquivalenzklasse durch ein einzelnes Element zu ersetzen, dazu führt, dass sich die Verteilung der Konkreta verschiebt. Im Beispiel entstanden nur noch kubische Pakte. Aus einer komplexen Verteilung der konkreten Pakete innerhalb einer Klasse wurde eine Punktverteilung. Multi-Level-Simulation muss diese Verteilung abbilden und dafür auch die Eindeutigkeit zwischen Abstrakta und Konkreta, wie sie in Abschnitt 3.10.4 gefordert wurde, aufgeben. Wir werden hierfür den Begriff der Verteilungskonformität verwenden.

Eine sinnvolle Multi-Level-Simulation darf die Variablenabstraktion aber auch nicht verletzen. Sie muss dafür sorgen, dass diese in jedem Zeitschritt erhalten bleibt. Würde bei einer Multi-Level-Simulation des Lieferkettenbeispiels aus einem abstrakten Paket mit Volumen 8 das konkrete Paket (1,1,1), würde der Zusammenhang zwischen beiden Repräsentationen des Paketes verloren gehen. Beide Simulationen würden beliebig voneinander divergieren. Um die Eindeutigkeit auflösen zu können, müssen wir daher zunächst eine Anforderung aufstellen, welche dennoch die Einhaltung der Variablenabstraktion sicherstellt. Wir werden dieses Konzept im weiteren Verlauf des Kapitels als Konsistenz bezeichnen.

Im Folgenden werden zunächst die Konsistenzanforderung und anschließend eine Anforderung zur Verteilungskonformität einer Multi-Level-Simulation abgeleitet.

### 4.3.1 Konsistenz

Der im Abschnitt 3.10 vorgestellte Lösungsansatz ist ein erster Versuch, Modelle unterschiedlicher Abstraktionsebenen zu koppeln. Auch wenn der Ansatz die dargestellten Probleme im Hinblick auf die Verteilung der Konkreta aufweist, garantiert er dennoch die Einhaltung der Variablenabstraktion. Wir sprechen davon, dass der Ansatz zu einer konsistenten Simulation beider Ebenen führt. Wir wollen die Konsistenz auch dann bewahren, wenn die Eindeutigkeit des Übergangs in die Detailsimulation aufgehoben wird. Ziel dieses Abschnittes ist es daher, diese Konsistenz näher zu beleuchten und in eine Anforderung an Multi-Level-Simulationen zu überführen.

Bereits in Davis (Davis, 1992) und Dangelmaier (Dangelmaier, 2004) wird eine Konsistenzbedingung für die Übergänge zwischen Abstraktionsebenen beschrieben. In beiden Arbeiten werden nicht nur Schnittstellen zwischen den Modellen hergestellt, sondern vollständige Modelle zur Laufzeit ausgetauscht. Entsprechend wird auch die Konsistenz bei beiden Autoren für den gesamten Zustandsraum gefordert. Abbildung 4.3 zeigt den diesen Arbeiten zugrundeliegenden Konsistenzbegriff. Der Zustand der Grobsimulation wird hier mit S bezeichnet. Der Zustand der Detailsimulation mit $S^*$. Zwischen beiden Zuständen wird eine Variablenabstraktion gefordert. Wird nun auf beiden Ebenen ein Simulationsschritt ($step$) ausgeführt, entstehen für den Zeitpunkt $t+1$ der neue Grobzustand $S'$ und für die Detailsimulation der neue Detailzustand $S^{*\prime}$.

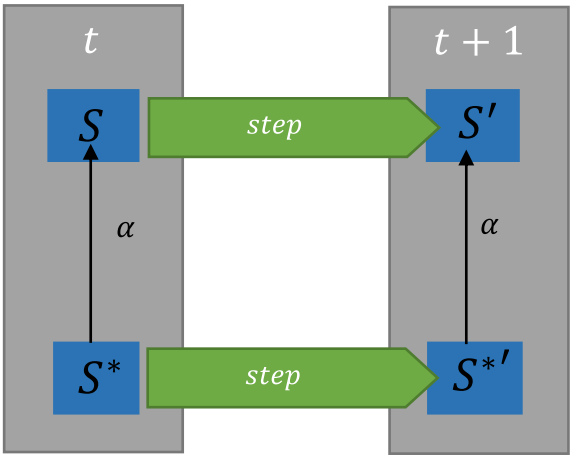



*Abbildung 4.3: Konsistenz nach Davis* (Davis, 1992), (Dangelmaier, 2004).

Beide Arbeiten fordern, dass auch zwischen diesen Zuständen wieder eine Variablenabstraktion besteht, also dass $\alpha(step(S^*)) = step(\alpha(S^*))$, wobei $step$ jeweils die Simulationsfunktion ist.

Die Adapter im Lieferkettenbeispiel verbinden jedoch nur einige der Variablen miteinander. Der Zustand innerhalb des Teilsystemmodells ist irrelevant, er wird nicht auf das Systemmodell

übertragen. Der Konsistenzbegriff von Davis ist daher für die im Rahmen dieser Arbeit angestrebte Multi-Level-Simulation zu restriktiv. Aus diesem Grund werden wir den Begriff für unsere Konsistenzanforderung auf einen weiter differenzierten Zustandsraum übertragen.

Die Belegung aller Variablen des groben Modells $S$ sowie aller Variablen des detaillierten Modells $S^*$ kann nach den Gesichtspunkten aus Abschnitt 4.1 in Inputs, Output und einen Rest aufteilen werden. Hierdurch erhalten wir:

$$S = I \cup O \cup R, \text{ sowie analog } S^* = I^* \cup O^* \cup R^*$$

Hierbei sind $I, O, I^*$ und $O^*$ wie in Abbildung 4.1 die Input- und Outputvariablen des betrachten Teilsystems innerhalb des System- beziehungsweise des Teilsystemmodells. $R$ sowie $R^*$ fassen alle übrigen Variablen beider Modelle zusammen.

Bei der Kopplung der Modelle mit Adaptern (siehe Abschnitt 3.10) sorgt $f$ vor einem Simulationsschritt dafür, dass zwischen den Belegungen von $I^*$ und $I$ eine Variablenabstraktion besteht. Zudem sorgt der Adapter $g$ dafür, dass in der Folgebelegung zum Zeitpunkt $t + 1$ die Belegungen $O'$ eine Variablenabstraktion von $O^{*\prime}$ ist. Beide Zusammenhänge werden in Abbildung 4.4 ebenfalls dargestellt.

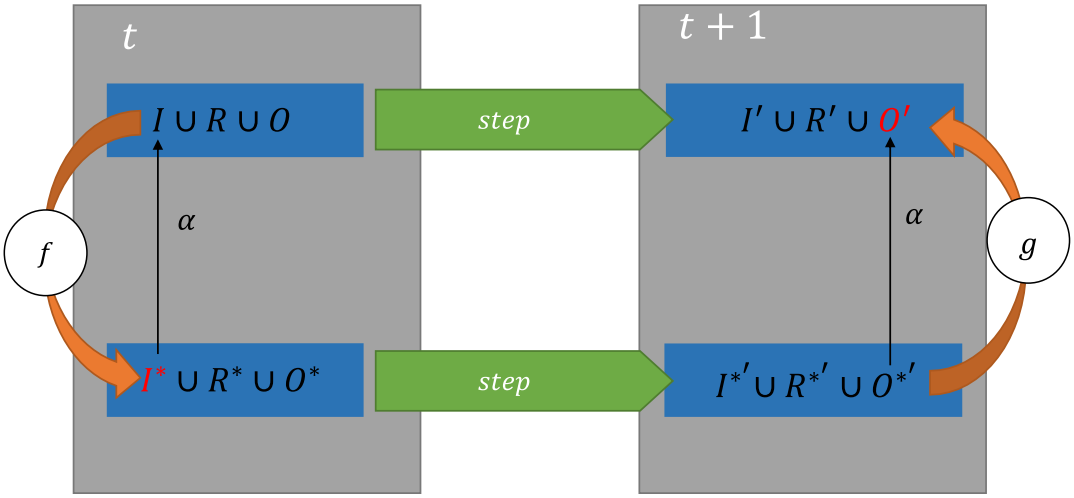



*Abbildung 4.4: Konsistenzanforderung an eine Multi-Level-Simulation.*

In einer späteren Version und vor dem Hintergrund des experimentellen Rahmens Zieglers, erweiterte Davis sein Konsistenzkriterium (Davis, 1998, S. 13). Er fordert nun nicht mehr die exakte Übereinstimmung der Zustände, sondern nur deren Übereinstimmung bis zu einem „ignorierbaren Fehler“. Dieser Fehler findet sich als Fehlertherm $\epsilon$ wieder. Wie groß er sein darf, muss der Modellierer im Kontext eines gegebenen experimentellen Rahmens festlegen.

Im Rahmen dieser Arbeit wird die Konsistenzanforderung an eine Multi-Level-Simulation daher wie folgt definiert:

$$\alpha(I^*) = I + \epsilon \ \wedge \alpha(O^{*\prime}) = O' + \epsilon$$

Die Abstraktion des detaillierten Inputs vor einem Simulationsschritt $\alpha(I^*)$ soll bis auf einen Fehler $\epsilon$ dem abstrakten Input $I$ entsprechen. Nach dem Simulationsschritt soll dann die Abstraktion des konkreten Outputs $\alpha(O^{*\prime})$ bis auf einen Fehler dem abstrakten Output entsprechen.

### 4.3.2 Verteilungskonformität

Wie bereits anhand der Kopplung des Beispiels mit Funktionen als Adaptern illustriert und diskutiert, erzeugt die Verwendung eines Adapters eine stark abweichende Verteilung der Konkreta von dem, was der Modellierer vorgesehen hat (Abschnitt 3.10.5).

Intuitiv würden wir von einer Multi-Level-Simulation erwarten, dass sich die modellierte Verteilung der Inputs auch in den generierten Konkreta wiederfindet. Aus dieser intuitiven Formulierung lässt sich eine weitere Anforderung an eine Multi-Level-Simulation entwickeln.

Die Menge der Konkreta (Belegungen von $I^*$), die sich bei einer isolierten Ausführung der Detailsimulation beobachten lässt, soll im Folgenden als $MODELLIERT^*$ bezeichnet werden. Die Menge aller Konkreta, die bei einer Multi-Level-Simulation an die Detailsimulation übergeben werden, soll im Folgenden als $GENERATED^*$ bezeichnet werden. Schließlich seien $MODELLIERT$ und $GENERATED$ die Mengen der zu $MODELLIERT^*$ bzw. $GENERATED^*$ gehörigen Abstrakta.

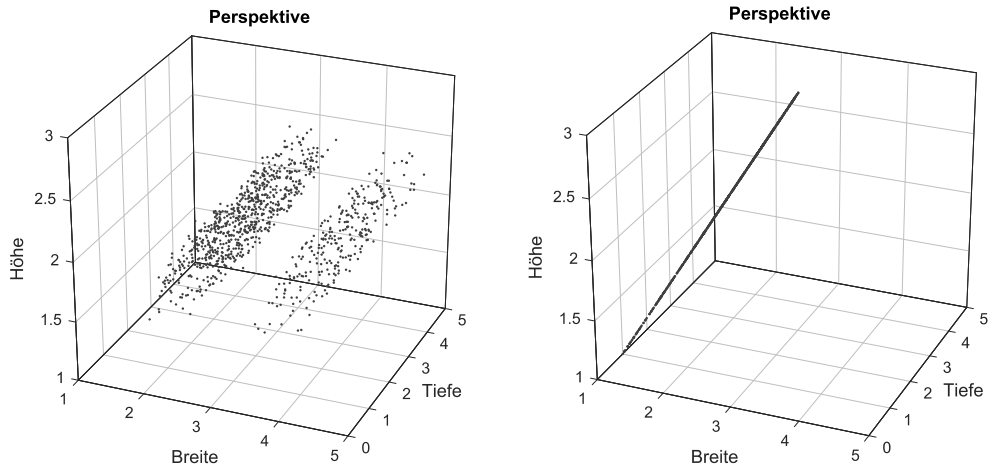



*Abbildung 4.5: Konkreta in $MODELLIERT^*$(links) und in $GENERATED^*$ (rechts).*

Abbildung 4.5 zeigt links $MODELLIERT^*$, also die Menge der Konkreta, die der Modellierer für die Inputs der Detailsimulation vorsieht. Rechts in der Abbildung befindet sich $GENERATED^*$, also die Menge der Konkreta, die während der gekoppelten Simulation durch den Adapter in Abschnitt 3.10 erzeugt wurden. Intuitiv sollten die Konkreta rechts so wie die Konkreta links verteilten sein. Dies ist offensichtlich nicht der Fall.

Ein erster Kandidat für die Anforderung ist daher die gleiche Verteilung beider Mengen zu fordern. Diese Intuition greift allerdings zu kurz. Hierbei würde die Verteilung der Konkreta isoliert betrachtet. In einer Multi-Level-Simulation beeinflusst jedoch auch die Verteilung der Abstrakta, die durch den Adapter in Konkreta umgewandelt werden, deren Verteilung.

Nehmen wir zum Beispiel an, dass nur Pakete mit der Größe 8 den Adapter erreichen. In diesem Fall müsste sich $GENERATED^*$ aus unterschiedlichen konkreten Paketen zusammensetzen, die alle das Volumen 8 besitzen. Egal wie gut der Adapter die Verteilung nachbildet, sie muss sich von der Verteilung von $MODELLIERT^*$ unterscheiden, da hier Pakete ganz unterschiedlicher Volumina enthalten sind. Trotzdem würden wir erwarten, eine Verteilung der Pakete mit Volumen 8, in $GENERATED^*$ wiederzufinden und eben nicht nur ein einzelnes Konkretum.

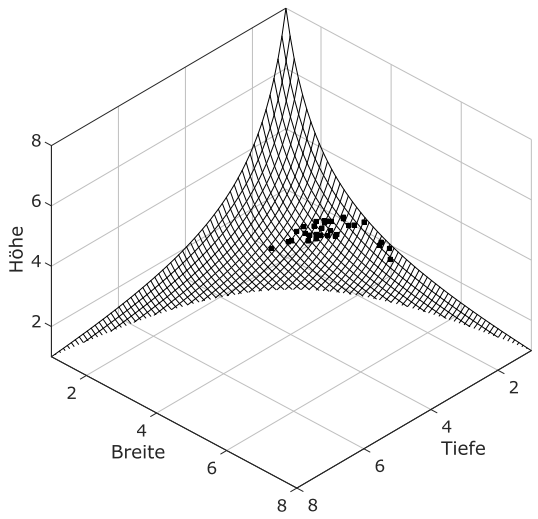



*Abbildung 4.6: $MODELLIERT^*$ für Konkreta mit Volumen zwischen 7,5 und 8,5.*

Abbildung 4.6 zeigt die Pakete aus dem Lieferkettenbeispiel, denen durch $\alpha$ ein Volumen zwischen 7,5 und 8,5 zugewiesen wird. Dazu findet sich in der Grafik auch ein Gitternetz, welches der gesamten Äquivalenzklasse von 8 entspricht. Intuitiv würden wir erwarten, dass die Konkreta, welche zu dem Abstraktum 8 generiert werden, ungefähr so verteilen sind, wie dies in der Abbildung der Fall ist.

Um diesen Zusammenhang der Konkreta in Abhängigkeit von den Abstrakta, auf deren Basis sie generiert werden, fassen zu können, eignen sich bedingte Wahrscheinlichkeiten. Betrachten wir hierfür den Wahrscheinlichkeitsraum $(\Omega, \Sigma, \mathrm{P})$. Dieser soll als wahre Verteilung der Abstrakta und Konkreta dienen.

Ein Ereignis $\omega \in \Omega$ besteht aus einem Paar $(a, k)$ wobei $a$ ein Abstraktum und $k$ ein Konkretum ist. Alle Kombinationen sind möglich. Auch Paare, in denen $a \neq \alpha(k)$. P weist jedoch dem Ereignis „a und k bilden keine Variablenabstraktion“ die Wahrscheinlichkeit 0 zu.

$$P(\{(a, k): a \neq \alpha(k)\}) = 0$$

Wegen der Additivität von Wahrscheinlichkeiten ist hierdurch auch die Wahrscheinlichkeit jedes einzelnen dieser Paare 0. Dies entspricht der Konsistenzanforderung aus Abschnitt 4.3.1.

Sei $M^*$ eine Zufallsvariable, welche Ereignisse dieses Wahrscheinlichkeitsraumes auf das Konkretum k projiziert. $MODELLIERT^*$ ist eine Stichprobe von $M^*$. Das heißt, wir nehmen an, dass der Modellierer die wahre Verteilung von $M^*$ der Konkreta kennt und dass sich diese in $MODELLIERT^*$ manifestiert.

Sei $M$ eine Zufallsvariable welche Ereignisse dieses Wahrscheinlichkeitsraumes auf das Abstraktum $a$ projiziert. $MODELLIERT$ ist eine Stichprobe von $M$.

$P(M^* = k|M = a)$ ist dann die Wahrscheinlichkeit, dass ein Konkretum $k$ auftritt, unter der Bedingung, dass das Abstraktum $a$ auftritt. Diese Wahrscheinlichkeit ist intuitiv die Wahrscheinlichkeitsverteilung der Konkreta einer Äquivalenzklasse $[k]_{\sim}$. Wir fordern, dass die Verteilung der generierten Konkreta in einer Multi-Level-Simulation dieser bedingten Wahrscheinlichkeitsverteilung folgt.

Sei also $(\Omega, \Sigma, \mathrm{P}_{\mathrm{Generated}})$ ein weiterer Wahrscheinlichkeitsraum mit den Zufallsvariablen $G$ und $G^*$ analog zu $M$ und $M^*$. $GENERATED$ und $GENERATED^*$ sind Stichproben von $G$ und $G^*$. Für eine sinnvolle Multi-Level-Simulation fordern wir schließlich:

$$P(M^* = k|M = a) = P_{Generated}(G^* = k|G = a)$$

Offensichtlich verletzt die Kopplung der Modelle mit Adaptern aus 3.10 dieses Kriterium. Hier wird allen Konkreta einer Äquivalenzklasse die Wahrscheinlichkeit 0 zugeordnet, bis auf ein einzelnes, welches mit 1 bewertet und somit immer ausgewählt wird. Im Lieferkettenbeispiel folgt der Adapter dem trivialen Fall einer Punkt- oder Dirac-Verteilung (auch deterministische Verteilung).

## 4.4 Zusammenfassung

Bei der Entwicklung komplexer Systeme wird häufig zunächst das Gesamtsystem in einem Grobentwurf ausgelegt, um später Feinentwürfe für Systemteile zu gestalten, welche schließlich wieder zu einem Gesamtsystem integriert werden. Für beide Entwürfe ist eine simulative Bewertung von Entwurfsalternativen verbreitet. Hierdurch entsteht der Rahmen für eine Abstraktionsebenen-übergreifende Simulation zwischen dem groben Modell des Gesamtsystems sowie detaillierten Modellen der Teilsysteme, welche in den Feinentwürfen betrachtet werden.

Zwischen den Modellen beider Ebenen können die drei wesentlichen Abstraktionsbeziehungen Approximation, Aggregation und Variablenabstraktion auftreten. Während Approximation und Aggregation statisch zwischen Modellelementen bestehen, bezieht sich die Variablenabstraktion auf Belegungen von Variablen während der Simulation beider Modelle. Entsprechend zentral ist die Variablenabstraktion für eine Multi-Level-Simulation beider Modelle.

Der Erhalt der Variablenabstraktion beim Ablauf einer Multi-Level-Simulation motiviert die Konsistenzanforderung. Das Lieferkettenbeispiel hat gezeigt, dass diese Konsistenz zwar notwendig, jedoch nicht hinreicht ist, um eine sinnvolle Multi-Level-Simulation zu beschreiben. Aus diesem Grund wurde die Konsistenz um die Verteilungskonformität ergänzt. Diese erlaubt es, die Eindeutigkeit zwischen Konkreta und Abstrakta aufzulösen und entsprechend der Intention des Modellierers eine komplexe Verteilung der Konkreta innerhalb einer Äquivalenzklasse abzubilden.

Im weiteren Verlauf werden wir den Algorithmus, der Abstrakta in Konkreta umwandelt (im Lieferkettenbeispiel f), mit $down$ und den Algorithmus, der Konkreta in Abstrakta umwandelt (im Lieferkettenbeispiel g), mit $up$ bezeichnen. Wir nutzen hierbei den Begriff Algorithmus und nicht etwa die Begriffe Funktion oder Abbildung, da diese eine Eindeutigkeit von $down$ implizieren würden, die hiermit gerade überwunden werden soll.

Im nächsten Kapitel werden bestehende Ansätze zur Abstraktionsebenen-übergreifenden Simulation in Hinblick auf diese beiden Anforderungen (Konsistenz und Verteilungskonformität) untersucht.

# 5 Stand der Forschung

Jedes Modell ist eine Abstraktion des Quellsystems. Modellierer müssen immer entscheiden, wie abstrakt ein Modell sein soll. Für unterschiedliche experimentelle Rahmen werden sie zu unterschiedlichen Entscheidungen gelangen. Aus diesem Grund beschäftigen sich Wissenschaftler immer wieder mit der Frage, wie mit dieser Entscheidung umgegangen werden kann.

Neben dem Ziel, Modelle unterschiedlicher Abstraktionsgrade zu verbinden, wie in dieser Arbeit, wird auch der Ansatz verfolgt, Modelle mit variablem Abstraktionsgrad zu erstellen. Tatsächlich kann das Beispiel aus Abschnitt 3 auch so aufgefasst werden, dass durch die Kopplung des groben Modells der Lieferkette mit dem detaillierten Modell der Kommissionierung der Abstraktionsgrad des Systemmodells punktuell erhöht wird.

In beiden Perspektiven ergibt sich jedoch dasselbe Problem. Es ist erforderlich, Übergänge zwischen den Zuständen der beiden Modelle zu definieren. Im Rahmen dieses Kapitels wird der Stand der Forschung auf dem Gebiet der Simulation von Modellen mit unterschiedlichen Abstraktionsstufen zusammengefasst. Dabei liegt der Focus insbesondere auf der Beschaffenheit der Übergänge in diesen Ansätzen sowie auf deren Erfüllung der Konsistenz- sowie Verteilungskonformitätsanforderung.

Den Ausgangspunkt dieser Betrachtungen stellt die Arbeit von Paul K. Davis (Davis, 1992) dar. Dieser prägt die Begriffe *cross-resolution model connection* für das Verbinden/Koppeln von Modellen unterschiedlicher Auflösung und den Begriff *variable-resolution modellig* für Modelle mit einstellbarer Auflösung. Zentrale Anwendungsdomäne seiner Arbeiten ist das Militär.

Unter Auflösung fasst Davis insgesamt sechs Teilaspekte zusammen. Die *entity resolution* beschreibt, wie feingranular die Entitäten (Objekte) der Simulation aufgeteilt werden, ob z.B. Armeen, Divisionen, Brigaden oder einzelne Soldaten modelliert werden. *attribute resolution* beschreibt, wie viele der Attribute dieser Entitäten jeweils abgebildet werden. *logical dependency resolution* beschreibt, wie detailliert die Beziehungen zwischen den Elementen modelliert werden. *process resolution* steht für die Detailfülle, in der das Verhalten der Entitäten abgebildet wird. Die *spatial resolution* beschreibt die räumliche und *temporal resolution* die zeitliche Auflösung eines Modells. Davis bezeichnet den Übergang von einem Detailzustand in einen Grobzustand ($up$) als Aggregation und den Übergang von einem Grobzustand zu einem Detailzustand ($down$) als Disaggregation.

Der Ansatz benennt mit *entity, attribute* und *spatial resolution* Dimensionen, welche zu einer Variablenabstraktion und somit zur Konsistenz- und Verteilungskonformitätsanforderung führen können. Das Konsistenzkriterium (siehe Abbildung 4.3) aus der Arbeit von Davis wurde

bereits vorgestellt. Dies ist tatsächlich nur die schwache Konsistenz bei Davis. In einer strengeren Version des Konsistenzkriteriums fordert er sogar die Existenz einer eindeutigen Disaggregationsfunktion, analog zu $down$.

Er räumt aber auch ein, dass dies im Allgemeinen nicht erreicht werden könne.

> *„Clearly, this level of consistency would be unusual, because aggregation [up] usually eliminates information."* (Davis, 1995)

Wie bereits angesprochen, erweitert Davis sein Kriterium später *(Davis, 1998).*

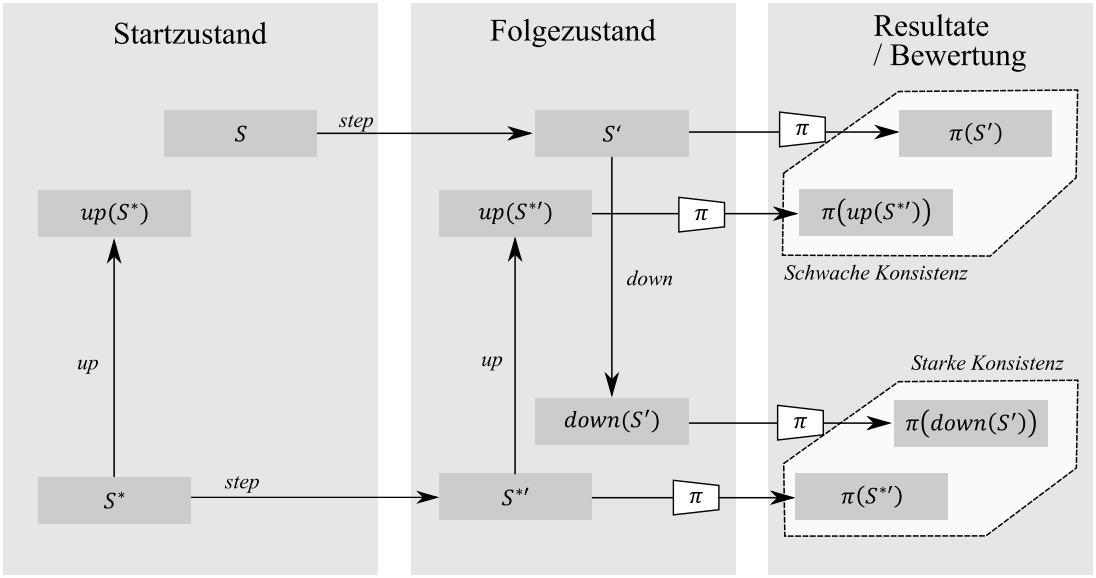



*Abbildung 5.1: Erweitertes Konsistenzkriterium nach Davis* (Davis, 1998).

Abbildung 5.1 zeigt dieses erweiterte Kriterium. Dabei lehnt sich die Struktur der Abbildung von Davis *(Davis, 1998)* an, nutzt aber die bereits eingeführten Notationen. Ausgangspunkte sind der Grobzustand $S$ und der Detailzustand $S^*$. Diese werden durch einen Simulationsschritt $step$ in die Folgezustände $S'$ bzw. $S^{*\prime}$ überführt. Zudem führt Davis nun den Projektionsoperator $\pi$ ein. Dieser reduziert einen Zustand auf die für die Bewertung des Zustandes notwendigen Eigenschaften. $\pi$ wird als Ausgabe einer Simulation, mit allen Nachverarbeitungsschritten, wie der Glättung oder statistischen Auswertungen, interpretiert.

*Schwache Konsistenz* ist nun eine Eigenschaft zwischen Projektionen von Grobzuständen. Hier muss, bis auf einen „ignorierbaren Fehler", $\epsilon$ gelten:

$$\pi(S') = \pi\left(up\left(S^{*\prime}\right)\right) + \epsilon$$

Also, dass die Projektion des neuen Grobzustandes der Aggregation ($up$) des neuen Detailzustandes entspricht.

*Starke Konsistenz* ist dann eine Eigenschaft zwischen Projektionen von Detailzuständen. Davis fordert:

$$\pi(S^{*\prime}) = \pi\big(down(S')\big) + \epsilon$$

Also, dass die Projekte des neuen Detailzustandes der Projektion des disaggregierten ($down$), neuen Grobzustandes entspricht. Für $down$ weist Davis zudem darauf hin, dass hier neue Information einfließen muss.

Die schwache Version des Kriteriums ist erneut äquivalent zu der in Abschnitt 4.3.1 diskutierten Konsistenzanforderung dieser Arbeit. Die starke Version beachtet jedoch nicht die im Rahmen dieser Arbeit vorgestellte Äquivalenzrelation zwischen verschiedenen Detailzuständen, welche zu der Verteilungsanforderung an Multi-Level-Simulationen führt. Stattdessen fordert Davis Gleichheit zwischen den Projektionen der Detailzustände. Insbesondere die Vorstellung einer eindeutigen Disaggregationsfunktion widerspricht der Verteilungsanforderung. Fast alle späteren Ansätze stützen sich direkt oder indirekt auf dieses starke Konsistenzkriterium, um eine geeignetes $down$ zu finden.

Davis weist bereits den Weg zu den in dieser Arbeit untersuchten Fragestellungen. Er liefert jedoch keine Lösung für das Abbilden der Zustände zwischen Modellen unterschiedlicher Ebenen, sondern konzentriert sich auf einen Ansatz zur variablen Modellierung von Verhalten (siehe Abschnitt 5.1.1).

Im Folgenden werden formale Konzepte und Frameworks vorgestellt, welche einen Domänen-unabhängigen Ansatz für die Modellierung und Simulation von Modellen unterschiedlicher Abstraktionsebenen liefern (Abschnitt 5.1). Daneben gibt es aber auch eine Vielzahl von Arbeiten, die motiviert durch ein konkretes, domänenspezifisches Simulationsszenario auf derartige Techniken zurückgreifen. Sie erarbeiten Lösungsvorschläge für konkrete Paare von Modellen aus diesen Domänen. Diese domänenspezifischen Ansätze werden in Abschnitt 5.2 diskutiert.

## 5.1 Formale Konzepte und Frameworks

Die Konzepte und Frameworks können in drei Kategorien aufgeteilt werden. Zum einen gibt es Ansätze, die Modelle verschiedener Abstraktionsstufen nutzen, um diese wechselseitig zu parametrieren (Abschnitt 5.1.1). So können etwa aus der Ausführung eines Modells Mittelwerte für ein abstraktes Modell abgeleitet werden. Dies entspricht dem Gedanken der Offline-Kopplung, wie er für das Lieferkettenbeispiel in Abschnitt 3.8 diskutiert wurde. Eine weitere Kategorie sind Ansätze, in denen Teile des Modells während der Simulation durch Teile mit einem anderen Abstraktionsgrad ausgetauscht werden (Abschnitt 5.1.2). Schließlich gibt es

Arbeiten, die es erlauben, Teile eines Modells auf mehreren Ebenen gleichzeitig zu simulieren und dabei die Zustände zu synchronisieren (Abschnitt 5.1.3).

### 5.1.1 Wechselseite Parametrierung von Modellen

Die Arbeiten dieser Kategorie zielen darauf ab, Modelle unterschiedlicher Abstraktionsstufe nebeneinander zu nutzen und Erkenntnisse, wie etwa die Verteilung einzelner Variablen, in ein anderes Modell zu übertragen.

Ein Beispiel hierfür ist das *integrated hierarchical variable-resolution* (IHVR) *modelling* (Davis, 1995). Hierbei geht es um die Erstellung von Modellfamilien. Der Ansatz wurde von Davis vor allem anhand von Beispielen aus dem Bereich der militärischen Simulation diskutiert. Eine Modellfamilie bezieht sich auf ein gemeinsames Quellsystem und die Modelle der Familie bilden dieses mit unterschiedlicher Auflösung (*resolution*) ab. Innerhalb einer IHVR Familie soll es möglich sein, die Modelle mit geringerer Auflösung durch die Modelle mit größerer Auflösung zu kalibrieren. Hierzu wird das Detailmodell ausgeführt und analysiert, um die Parameter des Grobmodells festzulegen.

Entsprechend konzentriert sich der Ansatz der IHVR darauf, ein Konzept für die Integration der Verhaltensbeschreibung auf Ebene der *Process Resolution* zu liefern. Hierbei wird das Verhalten, anders als in einer objektorientierten Perspektive, getrennt von dem Zustand und den Entitäten des Systems modelliert. Der erste Schritt, um aus zwei Modellen ein IHVR Modell zu erzeugen, besteht darin, alle Zustände und Entitäten sowie Variablennamen zwischen den Modellen zu vereinheitlichen. Somit bleibt die unterschiedliche *Process Resolution* als der relevante Unterschied zwischen den beiden Modellen übrig. Um diesen Unterschied zu überwinden, werden die Verhaltensbeschreibungen der ganzen Familie in einen Funktionsbaum überführt.

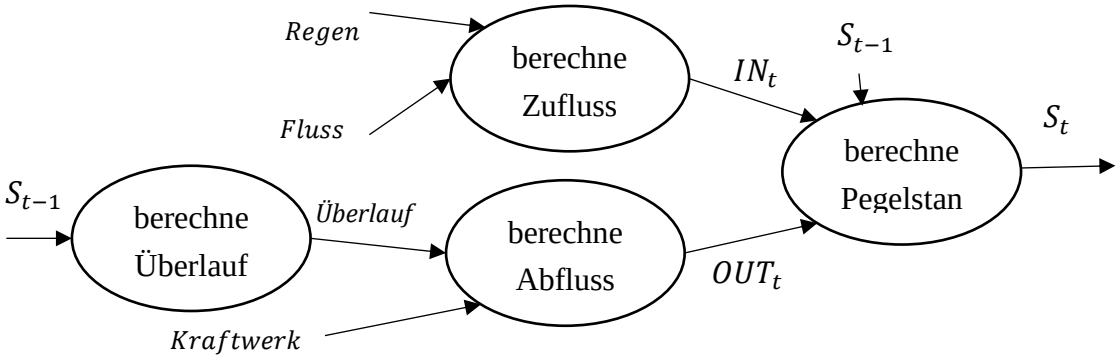



*Abbildung 5.2: Einfaches Beispiel für ein Model in einer IHVR.*

Die Knoten des Baumes sind dabei Funktionen. Diese nutzen Zustandsvariablen und Modellparameter als Eingabe und berechnen den neuen Wert einer Zustandsvariablen.

Abbildung 5.2 zeigt ein einfaches Beispiel für ein Modell einer IHVR-Familie. Im gezeigten Beispiel soll der Pegelstand $S_t$ eines Staudamms simuliert werden. Hierbei wird der Füllstand aus dem Füllstand im letzten Simulationsschritt $S_{t-1}$, dem Zufluss $IN_t$ und dem Abfluss $\mathrm{OUT_t}$ bestimmt. Der Zufluss wird aus dem Zulauf durch einen *Fluss* und durch *Regen* berechnet. Diese sind konstante Parameter des Modells. $\mathrm{OUT_t}$ wird aus dem Überlauf und den Abgängen über ein Kraftwerk berechnet. Wird ein bestimmter Pegelstand überschritten, verlässt zusätzliches Wasser über den Überlauf den Staudamm. Ein detaillierteres Modell der Familie könnte den konstanten Zufluss durch Regen dieses Modells durch eine wetterabhängige Berechnung ersetzten. Wird das Modell ausgeführt, kann der Durchschnitt des Zuflusses durch Regen über die Zeit als Parameter des weniger detaillierten Modells genutzt werden.

IHVR werden beispielsweise in (Davis, 2000) für die Simulation eines militärischen Invasions-Szenarios genutzt. In dem Szenario wird ein kleiner NATO-Staat (Blau) durch eine sich schnell fortbewegende, gepanzerte Streitmacht (Rot) überrannt. Blau steht eine Reihe von Präzisionsschlägen gegen die Übermacht zur Verfügung. Die Frage ist, ob dies ausreicht, um die Invasion zu stoppen. Ein detailliertes Modell modelliert, wie sich die gepanzerte Streitmacht durch das Gelände fortbewegt. Nur in offenem Terrain sind Präzisionsschläge möglich. Welchen Weg die Fahrzeuge wählen, wird wiederum von einer Vielzahl von Faktoren beeinflusst. Wie viele der Raketen von Blau ein Ziel in offenem Terrain antreffen, wird in diesem Modell analysiert. In einem groben Modell werden dann nur noch Wahrscheinlichkeiten für offenes Terrain als Parameter genutzt. Wie bei der Offline-Kopplung des Lieferkettenbeispiels (siehe Abschnitt 3.8), ist die Analyse manuell und setzt ein tiefes Verständnis beider Modelle voraus.

### 5.1.2 Dynamischer Austausch von Modellen

Eine weitere Gruppe von Arbeiten befasst sich damit, Teile eines Modells zur Laufzeit auszutauschen. Dies wird oft dadurch motiviert, dass nicht alle Teile des Modells über die gesamte Simulation in hohem Detailgrad benötigt werden. Durch den Austausch ist es möglich, Rechenressourcen zu sparen. Andersherum können auch bestimmte Teile des Modells plötzlich interessanter werden. Gibt es etwa in einer Verkehrssimulation ein Unfall, ist es ggf. interessant, die Straßen um den Unfall herum detailliert zu betrachten.

Um dies zu ermöglichen, muss der Zustand von dem deaktivierten Teilmodell auf das nun aktivierte Teilmodell übertragen werden. Da sie unterschiedliche Abstraktionsstufen realisieren, kann zwischen den Zuständen eine Variablenabstraktion (siehe Abschnitt 4.2.3) bestehen.

Die ***dynamic component substitution*** (DCS) (Rao, 2003) liefert einen solchen Ansatz. Hierbei wird jedes Modell als Ansammlung gekoppelter Komponenten verstanden. Jede Komponente entspricht einem Teilmodell. Die Komponenten tauschen Daten über Schnittstellen miteinander aus. Zudem können Komponenten atomar oder erneut aus Subkomponenten zusammengesetzt sein. Somit entsteht eine Hierarchie beliebiger Tiefe. Die DCS unterscheidet a priori (also bei Simulationsstart definierte) und reaktive (durch einen Trigger zur Laufzeit ausgelöste) Änderungen der Abstraktionsebenen der Modellteile. Um den Wechsel der Abstraktionsstufe zu realisieren, wird eine Komponente mit einer anderen ausgetauscht. Hierfür wird durch den Ansatz gefordert, dass die grobe und die detaillierte Komponente über identische Schnittstellen verfügen. Der Zustand muss bei Austausch der Komponenten mit geeigneten $up$ und $down$ Abbildungen übertragen werden. Diese werden durch den Modellierer manuell definiert.

Ein zentraler Beitrag der Arbeit ist die Quantifizierung von Fehlern und deren Ausbreitung durch das Komponentennetzwerk. Hierzu wird eine *error propagation library* (EPL) entwickelt. Diese definiert eigene Datentypen, welche neben dem eigentlichen Wert auch den Fehler fortschreiben. Die EPL ist durch Operatorenüberladung in die Implementierung der Komponenten integriert. Entsprechend werden die Komponenten in C++ unter Verwendung der EPL implementiert. Auch der mit $up$ und $down$ verbundene Informationsverlust wird mithilfe der EPL quantifiziert. Hierdurch wird es möglich, die Genauigkeit verschiedener Konfigurationen, sowie ganzer Simulationsläufe, gegenüberzustellen.

Die Komponenten werden vorab in isolierten Testsimulationen einer statistischen Analyse ihres Laufzeitverhaltens in Abhängigkeit von ihrem Input unterzogen. Hierdurch kann eine Abschätzung des erwarteten Rechenaufwandes gegeben werden (Rao, 2008, 2002). Zusammen mit der EPL kann so versucht werden, die - in Hinblick auf die Rechenressourcen - günstigste Konfiguration für eine Simulation zu finden, die einen definierten, höchsten erlaubten Fehler nicht überschreitet.

Der Ansatz wurde auf zwei Fallstudien mit Mobilfunknetzen (Rao, 2006) und digitalen Schaltungen (Rao, 2003) angewandt. Im ersten Fall ist die Abbildung zwischen den Zustandsräumen der Komponenten nicht mit einem Informationsverlust verbunden. Im zweiten Fall sind die Komponenten zustandslos, eine Abbildung ist nicht erforderlich.

Bei DCS werden die Übergänge zwischen den Zuständen manuell modelliert. Für die manuell definierten $up$ und $down$ werden keine Einschränkungen auferlegt. Wie hiermit jedoch die Konsistenz oder Verteilungskonformität erreicht werden können, wird nicht betrachtet.

**Multi-Resolution Modeling Space** ist ein Ansatz zur Spezifikation von gekoppelten Simulationsobjekten mit unterschiedlichen Abstraktionsebenen (Hong, 2013). Hierbei wird die Simulation in zwei Metaebenen getrennt.

Im sogenannten *simulation space* finden sich die *simulation object models* (SOM). Dahinter verbirgt sich jeweils ein Modell zusammen mit seinem Simulator. Die SOMs beschreiben eine Entität des Quellsystems auf einer bestimmten Abstraktionsebene mit einer bestimmten Auflösung. Zudem haben die SOMs Schnittstellen für Inputs und Outputs. Auch diese Schnittstellen sind mit einer Auflösung versehen. Mehrere SOMs können die gleiche Entität des Quellsystems beschreiben. Zudem kann es sein, dass eine Entität auf einer Abstraktionsebene durch ein einzelnes SOM und auf einer detaillierten Ebene durch mehrere SOMs modelliert wird. So kann eine Formation von Kampfjets auf einer groben Ebene als einzelnes SOM und auf einer detaillierten Ebene durch die drei SOMs einzelner Jets modelliert werden.

Die zweite Metaebene wird *multi-resolution space* genannt. Für jede Entität des Quellsystems gibt es hier eine sogenannte *multi-resolution model families*. Diese besteht aus einer Menge von *resolution object models* (ROM). Es ist immer nur eines der ROMs aktiv. Jedes dieser ROM ist mit den SOMs einer Abstraktionsebene verbunden. Die ROMs stellen ein Art Proxys für die SOMs des *simulation space* dar. Sie haben dieselben Schnittstellen wie ihre SOMs und leiten Nachrichten an die SOMs weiter.

Welche ROM einer Familie aktiv ist, kann sich dynamisch ändern. Der Ansatz setzt auf zwei Arten von Funktionen, um den Übergang zwischen den Abstraktionsebenen zu realisieren. Zum einen hat jede multi-resolution model familiy eine *resolution converter* $\psi$. Diese erhält ein ROM mit den zugehörigen SOMs einer Abstraktionsebene $i$ sowie eine Zielabstraktionsebene $j$. $\psi$ liest dann den Zustand der SOMs von $i$ und deaktiviert diese. Anschließend werden die SOMs der Ebene $j$ aktiviert. Die Funktion muss schließlich den Zustand der Ebene $i$ auf einen Zustand der Ebene $j$ abbilden und die SOMs damit initiieren. Dies entspricht entweder $up$ oder $down$. Wenn ROMs verschiedener Familien miteinander interagieren, ist es möglich, dass ihre Interfaces auf unterschiedlichen Abstraktionsebenen definiert wurden. Um diese zu überbrücken, werden in einem sogenannten *multi-resolution event interface* (MREI) Adapter zwischen den Interfaces definiert. Diese werden durch Funktionen $\omega_{i,j}$ realisiert, wobei $i$ und $j$ Abstraktionseben der Interfaces sind. Auch $\omega_{i,j}$ ist äquivalent zu $up$ (für $i < j$) und $down$ (für $i > j$). Zudem verfügt das MERI über ähnliche Konzepte wie Ptolemy, um Interfaces mit unterschiedlichen Zeitmodellen zu verbinden.

Sowohl $\psi$ als auch mit $\omega_{i,j}$ sind Funktionen. Dies bedeutet insbesondere, dass auch $down$ als Funktion modelliert wird und somit jedem Abstraktum immer ein einziges Konkretum zuordnet. Der Ansatz beschränkt sich darauf, leere Funktionen zu definieren, welche ein Modellierer ausgestallten muss (Hong, 2013, S. 30).

Prinzipiell kann durch die Annahme der 1-zu-1 Beziehung zwischen Konkreta und Abstrakta die Konsistenzanforderung erfüllt werden. Die Beschreibung von $down$ als Funktion erlaubt

jedoch nur eine Punktverteilung innerhalb der Äquivalenzklassen, sodass die Verteilungskonformität nur im trivialen Fall gewährleistet ist.

### 5.1.3 Synchronisation von Modellen

Bei den Arbeiten aus dem vorangegangenen Abschnitt ist immer nur eines der austauschbaren Teilmodelle aktiv. Die Schwierigkeit liegt in dem Initiieren der Teilmodelle bei einem Wechsel und darin, Adapter zwischen den Schnittstellen der Ebenen zu finden. In diesem Abschnitt werden Arbeiten dargestellt, welche diese Einschränkung aufheben und es erlauben, dass Entitäten gleichzeitig durch mehrere Teilmodelle simuliert werden. Die Schwierigkeit liegt nun darin, diese Zustände konsistent zu halten (Reynolds, 1997). Im weiteren Verlauf werden wir aufzeigen, dass die hierfür eingesetzten Konzepte starke Parallelen zu *up* und *down* aufweisen.

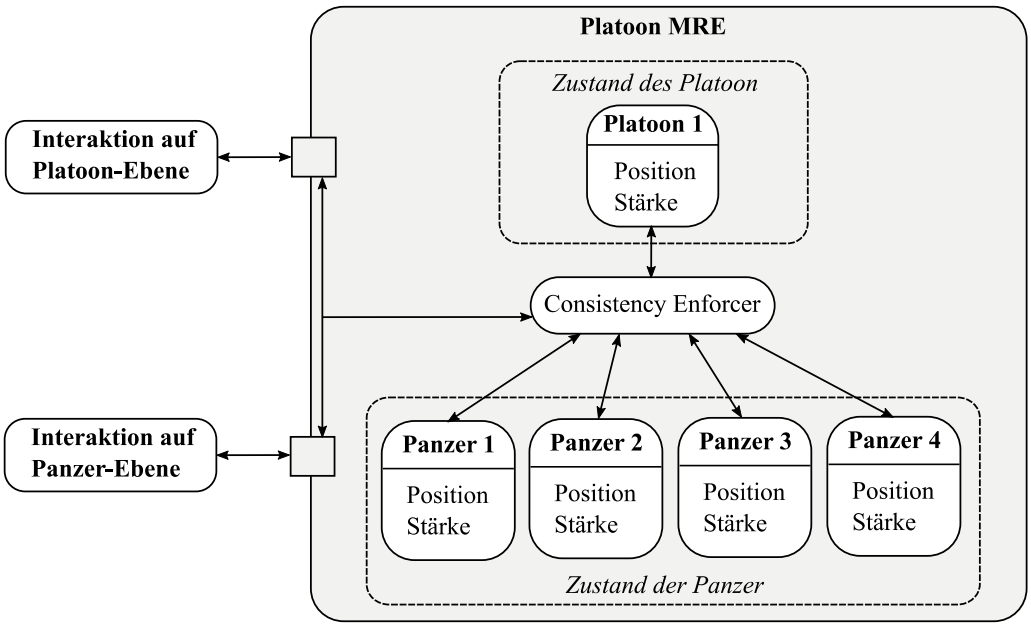



*Abbildung 5.3: multi-resolution entity in Anlehnung an Natrajan* (Natrajan, 1995, Abb. 3).

In seiner Dissertation beschreibt Natrajan einen Ansatz zur Simulation von sogenannten *multi-resolution entities* (MRE) (Natrajan, 2000). Hierbei werden alle Entitäten eines Systems durch die namensgebenden MREs modelliert. Eine MRE beschreibt dann diese Entität auf allen betrachteten Abstraktionsebenen. Auch dieser Ansatz stammt aus der Militärdomäne. Entsprechend sind bei Natrajan häufig militärische Einheiten Beispiele für MRE.

So kann etwa ein Platoon (Zug) als MRE modelliert werden (Natrajan, 1995). Das Beispiel von Natrajan wir in Abbildung 5.3 dargestellt. Die Abbildung lehnt sich an die entsprechende Abbildung von Natrajan an. Auf der groben Ebene verfügt das Platoon über einen *Zustand des Platoons*. Hier wird die Position des Platoons zusammen mit Eigenschaftswerten, wie z.B. einer Gesamtstärke, modelliert. Auf einer Detailebene wird dann das Platoon aus mehreren Panzern

modelliert. Diese haben jeweils eine eigene Position sowie einen eigenen Stärkewert. Diese Attribute bilden den *Zustand der Panzer*.

Je nachdem, ob eine Interaktion mit der MRE auf der Platoon-Ebene oder der Panzer-Ebene geschieht, wird eine andere Schnittstelle genutzt. Ein sogenannter *consistency enforcer* kümmert sich darum, dass Interaktionen an den Schnittstellen auf korrekte Weise an die beiden Zustände weitergeleitet werden. Zudem sorgt er dafür, dass beide Zustände zueinander konsistent bleiben. Hierzu nutzt der *consistency enforcer* die Abbildungen $f$. Diese Abbildungen sind immer invertierbar (Reynolds, 1997, Abschn. 7.1.2). $f$ und $f^{-1}$ stellen das Gegenstück zu $up$ bzw. $down$ dar. Sie erlauben es, den Zustand einzelner Ebenen zu „vergessen“, um Speicherplatz zu sparen, indem bestimmte Attribute nicht auf allen Ebenen vorgehalten werden, sondern bei Bedarf generiert werden.

Dies gilt nicht für die sogenannten Kernattribute. Diese müssen immer auf allen Ebenen gepflegt werden. Die Schwierigkeit hierbei ist es, Interaktionen auf einer Ebene konsistent auf die andere zu übertragen. Nehmen wir an, ein Panzer bewegt sich. Dies ist eine Interaktion auf Panzer-Ebene und führt zu einer Änderung des Zustands der Panzer. Die Änderung der Position des Platoons kann dadurch berechnet werden, dass der Schwerpunkt der Positionen der Panzer berechnet wird. Umgekehrt kann auch das Platoon seine Position durch eine Interaktion auf der Platoon-Ebene ändern. $Platoon.Positon+= \delta$. Um diese Änderung in eine Änderung des Panzerzustands zu überführen, ändert der *consistency enforcer* die Position aller dieser Panzer um denselben Wert. $Panzer_i.Postion+= \delta$. Da dies den Schwerpunkt korrekt verschiebt, bleibt die MRE konsistent.

Der Ansatz weist mehrere Schwachstellen auf. Die Invertierbarkeit erzwingt die Bijektivität von $f$. Hierdurch weist es jedem Abstraktum genau ein Konkretum zu, was die Verteilungsanforderung (siehe Abschnitt 4.3.2) gefährdet. In den Veröffentlichungen (z.B. (Natrajan, 1995), (Natrajan, 2000)) beschreibt Natrajan beispielhafte Abbildungen für Positionen oder Orientierungen. Hier kann eine eindeutige Abbildung $down$ erzeugt werden. Hierzu nimmt man an, dass die Panzer eine vorgegebene Formation einhalten. Wenn beispielsweise ein Platoon aus vier Panzern immer einer 20m langen Linie als Formation wählt, kann man aus Orientierung und Position des Platoons die Positionen und Orientierungen der einzeln Panzer eindeutig ableiten. Dies entspricht etwa der Annahme, dass die Pakete aus dem Beispiel immer kubisch sind, und widerspricht der Verteilungskonformitätsanforderung.

Auch in dem Platoon-Beispiel von Natrajan bleibt offen, was geschieht, wenn das Platoon Schaden nimmt. Tatsächlich ist dies eine Variablenabstraktion mit den bekannten Implikationen. Nehmen wir an, ein Angriff auf der Platoon-Ebene würde die Stärke um 25% reduzieren. Nun gibt es mehrere Möglichkeiten, diesen Schaden auf die Panzer aufzuteilen. So könnte der Schaden auf einen einzelnen Panzer angewandt werden. Dieser wäre dann zerstört

und könnte nicht mehr am Geschehen teilnehmen. Eine andere Abbildung würde den Schaden auf alle Panzer gleichmäßig aufteilen. Dadurch hätte jeder Panzer noch 75% seiner Stärke. Dies kann durch den Ansatz aber nicht gelöst werden. Durch die Forderung von Invertierbarkeit kann eine derartige Verteilung nicht einmal modelliert werden. Die vorgeschlagene Lösung besteht darin, diese Attribute als Kernattribute zu pflegen. Tatsächlich wird selbst die Position des Platoons bzw. der einzelnen Panzer in (Natrajan, 2000) als Kernattribut modelliert.

Auch der Ansatz, die Konsistenz zwischen den Zuständen der Kernattributen einer MRE dadurch zu erzielen, dass diese auf allen Ebenen gleichzeitig gepflegt werden, ist problembehaftet. Letztlich muss für jede MRE, und hier für jedes Kernattribut und für jede andere Abstraktionsebene, eine Abbildung definiert werden, welche Interaktionen mit dem Attribut auf die andere Ebene abbildet. Dies ist ein zusätzlicher Modellierungsaufwand. Zudem müssen Interaktionen mit Kernattributen immer auf allen Ebenen ausgeführt werden. Die Ersparnis, Berechnungen nur für die aggregierte Entität durchführen zu müssen, entfällt dadurch für Kernattribute. Bei genauerer Betrachtung stellt man fest, dass dies zu $up$ und $down$ äquivalente Adapter für die Schnittstellen erfordert. Ein Gedanke, welcher sich bei Baohong wiederfindet.

Baohong baut auf Basis des erweiterten Konsistenzbegriffes von Davis (siehe Abbildung 5.1) eine formale Spezifikation für *multi-resolution modeling* auf (Baohong, 2007). Dabei stützt er sich auf die formale Beschreibung für DE Modelle von Zeigler (Zeigler, 2019). Jedes Model beschreibt eine Entität des Quellsystems. In einer *multi-resolution model family* sind mehrere Modelle ein und derselben Entität mit unterschiedlichem Abstraktionsgrad zusammengefasst. Modelle dieser Familien können in einem sogenannten Modellnetzwerk gekoppelt werden. Es entsteht wieder ein Modell. Tatsächlich beweist Baohong sogar die Abgeschlossenheit der Menge aller Modelle bezüglich dieser Kopplung (Baohong, 2007). In einem Netzwerk entscheidet ein *resolution controller* anhand des Zustandes des Netzwerkes, welches der verschiedenen Modelle einer Familie aktiv ist. Ähnlich wie schon bei Natrajan können hier auch mehrere Module einer Entität gleichzeitig aktiv sein.

Um ein solches Netzwerk simulieren zu können, führt Baohong zwei Arten von Funktionen ein. So gibt es die Funktionen $Z$. Diese erfüllen die Aufgabe eines Konnektors zwischen Schnittstellen von Modellen verschiedener Familien. Zudem definiert Baohong Funktionen zwischen Modellen einer Familie. Betrachten wir diese zweite Klasse von Funktionen etwas genauer. Seien i und j zwei verschiedene Modelle einer Familie, dann ist $R_{i,j}: Y_i \rightarrow X_j$ eine solche Abbildung. Hierbei werden $Y_i$ (die Outputs des Modells i) auf $X_j$ (die Inputs des Modells $j$) abgebildet. Auch diese Funktionen nutzen ausschließlich die Interfaces der Modelle und haben keinen Zugriff auf deren inneren Zustand. Somit müssen alle Modelle einer Familie über ein geeignetes Interface verfügen, welches es ermöglicht, die Konsistenz zwischen ihren

Zuständen zu waren. Für diese Konsistenz definiert er zudem ein zwischen 0 und 1 normiertes Abstandsmaß (Baohong, 2005).

Auch für das Überbrücken der unterschiedlichen Auflösungen von Inputs müssen passende Interfaces existieren. Baohong lässt offen, ob die Konnektoren als Adapter zwischen den unterschiedlichen Ebenen genutzte werden oder ob es schlicht passende Interfaces geben muss. Da auch die Konnektoren Funktionen sind, unterliegen sie den gleichen Problemen wie die invertierbaren Funktionen $f$ von Natrajan (Natrajan, 1995). Auch die Vorstellung, dass es passende Interfaces an den Modellen gibt, ist problematisch. Nehmen wir, um dies zu erläutern, an, es gäbe ein derartiges zusätzliches Interface bei der detaillierten Kommissionierung. Dieses würde als Eingabe die abstrakten Pakete erhalten. Es wäre innerhalb des detaillierten Modells immer noch notwendig, einen Algorithmus $down$ zu definieren, der Konsistenz und Verteilung sicherstellt. Erneut werden keine Hilfestellungen hierfür gegeben.

## 5.2 Domänenanwendungen

Bei den bis hierher vorgestellten Ansätzen besteht immer der Anspruch eine allgemeingültige und domänenübergreifende Lösung zu finden. Eine große Zahl von Arbeiten beschäftigt sich damit, für bestimmte Anwendungsfälle und in definierten Domänen geeignete Lösungen zu finden. Fast immer wird dabei von einem Konsistenzbegriff ähnlich dem von Davis (Davis, 1998) ausgegangen. Im Folgenden wird eine Auswahl dieser punktuellen Lösungen vorgestellt.

### 5.2.1 Verkehrsdomäne

Eine Domäne mit einer Häufung dieser Ansätze stellt die Simulation von Straßenverkehr dar. Hier sind die verschiedenen Abstraktionsebenen bereits lange ein von der wissenschaftlichen Gemeinschaft gewürdigter Aspekt. Entsprechend liegt es nahe, zu versuchen, die Brücke über die Ebenen hinweg zu schlagen.

So stellt Gutlein (Gutlein, 2018) ein Framework für hybride Verkehrssimulation vor. Hierbei soll eine Integration in drei Dimensionen erfolgen. Zunächst eine Integration verschiedener Modellierungsdomänen, wie etwa Umgebungs-, Bevölkerungs-, Verkehrs- oder Netzwerkmodellen. Die Kopplung dieser Domänen wird in der Arbeit als hybride Simulation bezeichnet. Hinzu kommt aber auch eine Integration verschiedener Abstraktionsebenen. Hierbei wird zwischen vier Ebenen (*macro*, *meso*, *micro* und *submicro*) unterschieden. Die Kopplung verschiedener Abstraktionsebenen wird als Multi-Level-Simulation bezeichnet. Hierbei werden besonders interessante Teile eines Verkehrsnetzes, wie etwa eine viel befahrene Straße, in einer der detaillierten Ebenen modelliert und mit einem groben Modell der restlichen Stadt verbunden. Perspektivisch sollen dabei auch Fahrzeuge zu Gruppen aggregiert werden. Dies würde einer Variablenabstraktion entsprechen (siehe Abschnitt 4.2.3). In der vorgestellten

Version ist diese jedoch nicht implementiert. Auch entsprechende Abbildungen $up$ und $down$ sind noch offen. Entsprechend betrachtet das Framework weder Konsistenz noch Verteilung.

Claes beschreibt einen Ansatz für eine dynamische Verkehrssimulation (Claes, 2009). Es werden zwei unterschiedliche Modelle der Straßenabschnitte betrachtet. In einem Warteschlangenmodell werden die Fahrzeuge nach dem Last-in-First-out-Prinzip modelliert. Für jedes eintretende Fahrzeug $i$ wird, basierend auf seiner Durchschnittsgeschwindigkeit und der Länge des Straßenabschnitts, eine früheste Zeit des Verlassens der Warteschlange ($t_i$) bestimmt. Das Fahrzeug $k$ am Kopf der Warteschlange verlässt diese, sobald $t_k$ erreicht ist. Alle weiteren Fahrzeuge müssen ggf. in der Warteschlange verweilen, auch, wenn ihr $t_i$ bereits erreicht ist. In einem zweiten kontinuierlichen Modell werden die Fahrzeuge mit eindimensionaler Position $p_i$ auf der Straße modelliert. In jedem Simulationsschritt geben alle Fahrzeuge eine Wunschentfernung an, welche sie zurücklegen wollen. Diese basiert auf ihrer Geschwindigkeit. Die Umgebung bewegt die Fahrzeuge um diese Entfernung. Falls ein vorausfahrendes Fahrzeug im Weg ist, wird die Entfernung entsprechend reduziert. Das Warteschlangenmodell wird hierbei als weniger rechenintensiv bewertet. Dafür ist das kontinuierliche Modell bei mittleren Verkehrsdichten präziser. Der Simulationsansatz besteht nun darin, je nach Verkehrsdichte, zwischen den Modellen der einzelnen Straßen zu wechseln. Hierbei wird eine einfache Zustandsabbildung zwischen dem Warteschlangenmodell und dem kontinuierlichen Straßenmodell genutzt. Diese bildet $t_i$ und $p_i$ eindeutig ineinander ab. Auch der Übergang zwischen Straßenabschnitten auf den unterschiedlichen Abstraktionsebenen ist trivial. Verlässt ein Fahrzeug das kontinuierliche Modell, wird es an das Ende der Warteschlangen einer Folgestraße gesetzt. Verlässt ein Fahrzeug eine als Warteschlange modellierte Straße, wird es an der Startposition einer sich anschließenden, kontinuierlichen Straße platziert. Die Arbeit erfüllt die Konsistenzanforderung, erzwingt aber eine 1-zu-1 Beziehung zwischen Abstrakta und Konkreta, was der Verteilungskonformität widerspricht.

Auch in (Sewall, 2011) werden Verkehrssimulationen unterschiedlicher Abstraktionsebenen miteinander verbunden. Zum einen, eine detaillierte Simulation einzelner Fahrzeuge. Diese besitzen eine Position auf der eindimensionalen Straße, eine Länge sowie eine Geschwindigkeit. Das Verhalten des Fahrers wird durch einen Agenten repräsentiert. Dieser wählt z.B. die Geschwindigkeit des Fahrzeugs. Zum anderen, ein gröberes DG-Modell aus diskreten Straßenabschnitten. Diese Abschnitte weisen alle eine kontinuierliche Verkehrsdichte sowie eine kontinuierliche Flussgeschwindigkeit auf. Das Modell ähnelt dabei der Simulation einer Flüssigkeit in einem Rohrleitungssystem. Einzelne Fahrzeuge werden nicht modelliert. Welche Bereiche eines großen Straßennetzes mit welchem Modell untersucht werden, kann dynamisch gewechselt werden. Zwischen der abstrakten Verkehrsdichteverteilung des DG-Modell und der Menge von Fahrzeugen des agentenbasierten Modells besteht eine Variablenabstraktion. Verschiedene Mengen diskreter Fahrzeuge erzeugen die gleiche Dichte.

Um zwischen den Modellen wechseln zu können, muss $up$ aus einer Menge von Fahrzeugen mit Position und Geschwindigkeit die Verkehrsdichte aggregieren. Dies ist eindeutig möglich. Für $down$ wird in der Arbeit ein Generator modelliert. Dieser nutzt die Verkehrsdichte als Wahrscheinlichkeitsdichte eines Poisson Prozesses. Dieser stochastische Prozess wird eigentlich für die Modellierung zeitlicher Ereignisse eingesetzt. Hier wird er eingesetzt, um entlang der eindimensionalen Straße zufällig Fahrzeuge entsprechend der angegebenen Verkehrsdichte zu generieren. Der Ansatz stellt sicher, dass die generierten Fahrzeuge (bei hinreichend häufiger Wiederholung der Generierung) die Dichte des DG-Modells treffen. Das vorgestellte $down$ kann als ein Beispiel für ein manuell erstelltes $down$ aufgefasst werden, welches die Verteilungsanforderung erfüllt. Somit erfüllt dieser domänenspezifische Ansatz mit dem vorgestellten $down$ beide Anforderungen.

***multi-level agent-based simulation*** ist ein Ansatz für die agentenbasierte Simulationen in urbanen Regionen (Navarro, 2011). Hierbei werden Agenten genutzt, um die Passenten zu modellieren. Jeder Agent verfügt über eine Reihe von Attributen. Navarro unterteilt diese Attribute in physikalische und psychologische Attribute. Um Rechenzeit zu sparen, werden einige der Agenten dynamisch zu Gruppen zusammengefasst. Eine feste Zuordnung, welche Agenten zu einer Gruppe gehören, gibt es nicht. Hierfür wird ein Abstandsmaß definiert. Dieses berücksichtig die Position, und die Ähnlichkeit der Agenten, und fasst nahe, ähnliche Agenten zusammen. Um eine solche Gruppe zu aggregieren, werden die einzelnen Agenten gelöscht und durch eine Gruppe ersetzt. Diese verfügt über dieselben Attribute wie die einzelnen Agenten. Der Ansatz kann um eine Zwischenebene, sowie einen Ansatz für eine unterschiedliche Prozessauflösung (vgl. IHVR) erweitert werden (Navarro, 2013). Für jedes Attribut muss nun eine Aggregationsfunktion, äquivalent zu $up$ gefunden werden. Im umgekehrten Fall wird ein $down$ benötigt. Auch Navarro nutzt das schwache Konsistenzkriterium von Davids (Davis, 1998) für seinen Ansatz. Zudem schlägt er vor, einen Konsistenzfehler als Kriterium für die Auswahl von $up$ zu nutzen. Hierfür kann aus einer Vielzahl von definierten $up$ Kandidaten derjenigen ausgewählt werden, welcher den geringsten Konsistenzfehler erzeugt. Im Ausblick der Arbeit von Navarro wird vorgeschlagen, hierfür maschinelles Lernen einzusetzen (Navarro, 2011). Für $down$ wird eine Memory-Funktion vorgeschlagen. Diese speichert bestimmte Eigenschaften des detaillierten Zustandes (der vielen Agenten) und erlaubt, einen eindeutigen, „ähnlichen“ Detailzustand zu reproduzieren. Betrachten wir ein Beispiel aus (Navarro, 2011). Bei Ressourcen wie Bargeld stellt die Summenbildung einen sinnvollen Kandidaten für $up$ da. Haben drei Agenten die Geldmenge $A$, $B$ und $C$, würde $up$ daraus die Summe $S$ machen. Eine Memory-Funktion könnte das Verhältnis $\frac{A}{S}$, $\frac{B}{S}$ und $\frac{C}{S}$ speichern und $down$ dann wie folgt definierten:

$$down(y) = \left(\frac{A}{S}, \frac{B}{S}, \frac{C}{S}\right)^T * y$$

Ein Nachteil dieser Memory Funktion ist, dass sie für jede aggregierte Gruppe gespeichert werden muss. Navarro schlägt vor, diese nach einer gewissen Zeit zu vergessen. Werden Gruppen ohne Memory-Funktion wieder in einzelne Agenten aufgelöst, schlägt er eine zufällige Belegung der Attribute vor. Dies stellt auch die einzige Möglichkeit da, wenn die Simulation mit einer Gruppe startet. In diesem Fall kann es keine Memory-Funktion geben. Problematisch bei zufälliger Wahl der Attribute bleibt die Konsistenz. Die Autoren gehen nicht auf dieses Problem ein. Zudem muss der Modellierer diese Memory-Funktion für jedes Attribut definieren, ebenso, wie einen Mechanismus, sie zu generieren.

Die Memory-Funktion stellt erneute eine Möglichkeit da, um eine 1-zu-1-Beziehung für $down$ zu erzeugen. Dies erfüllt die Verteilungskonformität jedoch erneut nur im trivialen Fall einer Punktverteilung.

### 5.2.2 Weitere Domänen

**Smart Grid.** Mosaik ist ein Python-basiertes Simulationsframework, welches es erlaubt, Modelle aus der Domäne der intelligenten Stromnetze zu koppeln (Schütte, 2011). Hierzu müssen die Modelle eine API umsetzen. Nach dem Start melden sich die Modelle an einen Masteralgorithmus an. Dieser orchestriert die Modelle gemäß einer Szenariodefinition. Innerhalb dieser Szenariodefinition kann der Modellierer auf eine Sammlung von domänenspezifische Datentypen für die Schnittstellen zwischen den Modellen zurückgreifen. Über Ihre API müssen sich alle Modelle in eine DT-Taktung mit globaler Schrittweite fügen. Die gekoppelte Simulation folgt somit ebenfalls dem DT-Zeitmodell.

In seiner Dissertation (Schütte, 2013) formuliert Schütte zudem die Anforderung, dass es mit Mosaik möglich sein muss, Modelle unterschiedlicher Abstraktionsstufen zu koppeln. Als Beispiel hierfür nennt er die Verbindung des groben Modells eines Leitungssystems mit dem detaillierteren Modell eines Elektrofahrzeuges. Das Leitungssystem modelliert den Energiefluss mit nur einer Phase. Das Elektrofahrzeug modelliert diesen jedoch mit drei getrennten Phasen. Es besteht eine Variablenabstraktion. Um diese Anforderung zu erfüllen, kann der Modellierer bei der Beschreibung eines Szenarios eine vordefinierte Mapping-Funktion auswählen, oder selbst eine solche Funktion spezifizieren. Im Beispiel ist $up$ die Summe der Phasen und $down$ z.B. eine gleichmäßige Verteilung des Energieflusses auf alle drei Phasen. Zwischen $up$ und $down$ wird keine Konsistenz gefordert.

**Materialfluss**. Ein weiteres in der Literatur beschriebenes Anwendungsgebiet für eine dynamische Simulation auf unterschiedlichen Abstraktionsebenen ist die ereignisbasierte Materialflusssimulation. Hierbei werden die Produktions- und Logistikprozesse, beispielsweise einer Fabrik, als DE-Modell simuliert. Es gibt Ansätze, im Rahmen der sogenannten digitalen Fabrik (Weisser, 2008), diese Simulation als Teil einer virtuellen 3D Welt für den Nutzer zu

visualisieren. Zudem kann der Nutzer mit den einzelnen Maschinen der Fabrik interagieren und so direkt was-wäre-wenn-Szenarien durchspielen. Um für den Nutzer eine angenehme und flüssige Visualisierung liefern zu können, unterliegt die Simulation in diesen Fällen Echtzeitanforderungen. In diesem Kontext untersucht Dangelmaier (Dangelmaier, 2004) einen Ansatz, in dem Teile des Modells dynamisch durch abstraktere Teile ausgetauscht werden. Hiermit ist ein Performancegewinn verbunden. Die Position und Perspektive des Nutzers innerhalb der virtuellen Welt entscheidet darüber, wo detaillierte Modelle genutzt werden. Teile der Fabrik, die der Nutzer gerade nicht sieht und die weit entfernt von ihm sind, werden durch abstraktere Teilmodelle ersetzt. Hierbei werden Gruppen von verbundenen Maschinen zu einzelnen Maschinen abstrahiert. Welche Gruppen zusammen gefasst werden, kann Anhand eines Regelwerkes ermittelt werden (Huber, 2009).

Um die Simulation konsistent zu halten, müssen bei dem Wechsel zwischen beiden Ebenen die Zustände der neu aktivierten Ebene initiiert werden. Die Abbildung des Zustands der Gruppe zu einer einzelnen Maschine ist dabei äquivalent zu $up$, wärend der umgekehrte Vorgang äquivalent zu $down$ ist. Es existieren verschiedene Ansätze, wie beide Algorithmen umgesetzt werden können. Zum einen schlägt Dangelmaier mit „Resimulation“ eine Strategie vor, in der alle Ereignisse bis zu einem bestimmten Zeitpunkt in der Vergangenheit auf der nun aktivierten Einzelmaschine bzw. Gruppe wiederholt werden. Das Problem hierbei ist, dass zum einen alle betreffenden Events gespeichert werden und dass die Rechenzeit für deren Abarbeitung nun in kurzer Zeit bei einem Wechsel notwendig wird. In einer zweiten Strategie wird hierfür ein manuell definierter Algorithmus $up$ bzw. $down$ vorgesehen.

Für diese konkrete Domäne liefert Dangelmeier zusammen mit Huber einen Vorschlag für $up$ und $down$ (Huber, 2011). Hierbei richtet sich Huber nach dem schwachen Konsistenzkriterium von Davids (siehe Abbildung 5.1). Aus der Domäne heraus kann tatsächlich eine in weiten Teilen eindeutige $down$ Funktion definiert werden. Mit gewissen Einschränkung ist es möglich, z.B. die Fertigungsaufträge der Einzelmaschine eindeutig auf eine der Maschinen der Gruppe abzubilden. Für die Fertigungsaufträge der Einzelmaschine ist bekannt, wann sie diese verlassen werden. In der Gruppe sind die Bearbeitungsdauern festgelegt und bekannt. Hiermit ist die Maschine innerhalb der Gruppe, sowie die hier übrige Bearbeitungsdauer, eindeutig bestimmt. Dies gilt jedoch nur dann, wenn alle Gruppen aus Maschinen in einer Reihe zusammengesetzt sind. Gibt es Verzweigungen, ist eine Verteilung im Sinne der Verteilungsanforderung notwendig. Dieser Fall wird jedoch nicht betrachtet. Auch weitere Elemente des Zustandes der Maschinen werden von $down$ nur betrachtet, wenn eine eindeutige Funktion gefunden werden kann. Somit ist erneut die Konsistenzanforderung erfüllt. Die Verteilungskonformität ergibt sich jedoch nur bei eine Punktverteilung.

## 5.3 Zusammenfassung

Die vorgestellten Arbeiten zeigen, dass die Idee der Simulation über verschiedene Abstraktionsebenen hinweg weit verbreitet ist. Von zentraler Bedeutung ist hierbei das Konsistenzkriterium von Davis (Davis, 1998). In seinem Kern legt das starke Konsistenzkriterium eine 1-zu-1-Beziehung zwischen Abstrakta und Konkreta fest. Dies wird als erforderlich für die Existenz von $down$ angesehen. Gibt es eine 1-zu-N-Beziehung, werden zusätzliche Einschränkungen für die Äquivalenzkassen gefunden, bis wieder 1-zu-1 gilt. Dieser Ansatz widerspricht im Kern der Verteilungskonformität (Abschnitt 4.3.2). Nahezu allen dargestellten Arbeiten beziehen sich auf das Konsistenzkriterium von Davis oder erarbeiten selbst ähnliche Vorstellungen.

Zudem wird die Aufgabe, rettenden Disaggregationsfunktionen zu finden, welche diese 1-zu-1-Relation herstellen, oft dem Modellierer zuteil. Hilfestellungen hierfür, die über das Konsistenzkriterium hinaus gehen, bietet keiner der betrachteten Ansätze. Dies ist auch der Grund, weshalb es eine Vielzahl von Arbeiten gibt, welche für konkrete Domänenproblem nach geeigneten $up$ und $down$ Algorithmen suchen (vgl. Anschnitt 5.2). In der Regel, in dem sie nach Einschränkungen für die Ausgabe von $down$ suchen, sodass diese eindeutig wird. Sicherlich gibt es Fällen, in denen dies gelingt. Allerdings zeigt das vorgestellt Beispiel in Kapitel 3, dass dies nicht immer der Fall ist.

Einzige Ausnahme stellt Ansatz von Sewall für zwei Modelltypen aus der Verkehrsdomäne da (Sewall, 2011). Er definiert in seiner Arbeit ein nichtdeterminiertes $down$, welches einer aufwendig konstruierten Verteilung folgt. Der Ansatz ist dabei spezifisch für die beiden Modelltypen und naturgemäß nicht übertragbar auf Äquivalenzklassen mit einer anderen Verteilung.

Einen allgemeinen, domänenunabhängigen Ansatz, der die Verteilungskonformitätsanforderung erfüllt, und der keine triviale Verteilung innerhalb der Äquivalenzklassen voraussetzt, gibt es nicht.

# 6 Ziel und Forschungsfragen

In Kapitel 5 wurde aufgezeigt, dass es keinen allgemeinen Ansatz für Abstraktionsebenen-übergreifende Simulation gibt, der in der Lage ist, die Konsistenz- und die Verteilungsanforderungen (4.3.1 und 4.3.2) zu erfüllen. Zudem erfordern fast alle Ansätze die manuelle Definition der Übergänge zwischen den Ebenen. Dieser zusätzliche Modellierungsaufwand birgt die Gefahr, dass die Abstraktionsebenen-übergreifende Simulation ihren Vorteil gegenüber der vollständigen Detailsimulation (vgl. 3.9) verliert.

Daher hat diese Arbeit zum Ziel, ein Multi-Level-Simulationskonzept zu finden,

- welches die Konsistenz- und die Verteilungsanforderungen (4.3.1 und 4.3.2) erfüllt und
- einen möglichst kleinen, zusätzliche Aufwand vom Modellierer fordert.

Um dieses Ziel zu erreichen, wird der Lösungsansatz verfolgt, die Übergänge zwischen den Simulationen ($up$ und $down$) mithilfe von Methoden des maschinellen Lernens und anhand von Beispielen zu ermitteln.

Es werden die folgenden Forschungsfragen untersucht:

1. **Wie kann eine Architektur für lernende Multi-Level-Simulation aussehen?** Diese Architektur muss insbesondere die Konsistenzanforderung aus 4.3.1 sicherstellen. Zudem muss sie einen Rahmen für das Lernen von $up$ und $down$ liefern und dabei insbesondere festgelegt, wie $up$ und $down$ beschrieben werden, sodass sie im Rahmen der Architektur eingesetzt werden können.
2. **Welche Methode des maschinellen Lernens eignet sich zur Bestimmung von $up$?** Hierzu muss die Methode anhand weniger Beispiele die Variablenabstraktion $\alpha$ nachbilden.
3. **Welche Methode des maschinellen Lernens eignet sich zur Bestimmung von $down$?**
   Hierzu muss die Methode aus den modellierten Inputs die bedingte Wahrscheinlichkeit aus der Verteilungsanforderung 4.3.2 nachbilden.
4. **Fügen sich die Lernverfahren für Algorithmen $up$ und $down$ tatsächlich mit der Architektur zu einer Multi-Level-Simulation zusammen?** Neben der isolierten Evaluation der beiden Lernverfahren ist hierzu eine Evaluation der lernenden Multi-Level-Simulation im Rahmen von Evaluationsszenarien notwendig.

# 7 Multi-Level-Simulationen

Dieses Kapitel führt die formale Spezifikation für die Architektur einer gekoppelten Simulation von Modellen mit unterschiedlichen Abstraktionsstufen ein. Die Architektur erfüllt die Konsistenz- sowie die Verteilungskonformitätsanforderungen und leitet Anforderungen an $up$ und $down$ ab. Der Ansatz wird im weiteren Verlauf der Arbeit als Multi-Level-Simulation bezeichnet.

## 7.1 Simulationsmodelle

Analog zur Formalisierung von Ptolemy (Tripakis, 2013) werden Simulationsmodelle in dieser Arbeit als Menge von Variablen definiert. Diesen Variablen werden mithilfe einer Belegungsfunktion Werte aus einem Werteuniversum zugewiesen.

Sei VARIABLE die Menge aller Variablennamen und DOMAIN das Universum aller Werte. Dann ist eine Belegung von Variablen eine Funktion x, welche jeder Variable einer Teilmente $\mathrm{V} \subseteq \mathrm{VARIABLE}$ einen Wert aus DOMAIN zuweist.

$$x: V \rightarrow DOMAIN$$

Die Menge aller Belegungen einer Menge von Variablen V wird im weiteren Verlauf der Arbeit mit $\widehat{\mathrm{V}}$ bezeichnet.

$$\widehat{\mathrm{V}} \coloneqq \{\mathrm{x} \in (\mathrm{V} \rightarrow \mathrm{DOMAIN})\}, \text{wobei } \mathrm{V} \subseteq \mathrm{VARIABLE}$$

Die Variablen eines Simulationsmodells teilen sich in zwei Arten auf. Zum einen gibt es Zustandsvariablen und zum anderen Eingabevariablen. Zustandsvariablen beschreiben, ihrem Namen entsprechend, den inneren Zustand des Simulationsmodells. Eingabevariabeln modellieren externe Stimuli und Einflussgrößen der Umgebung des betrachteten Systems. Entsprechend können die Menge aller Eingabevariablen $INPUT$ und die Menge aller Zustandsvariabeln $STATE$ als disjunkte Teilmengen von $VARIABLE$ definiert werden.

$$INPUT, STATE \subseteq VARIABLE \wedge$$

$$INPUT \cap STATE = \emptyset \ \wedge \ INPUT \cup STATE = VARIABLE$$

Basierend auf diesen beiden Variablenuniversen kann nun das Universum aller Simulationsmodell definiert werden. Ein Simulationsmodell besteht aus einer Teilmenge der Eingabevariablen, einer Teilmenge der Zustandsvariablen sowie einer Belegung einer Teilmenge der Zustandsvariablen, welche als Startzustand fungiert. Entsprechend setzt sich das

Universum aller möglichen Simulationsmodell aus Elementen, den entsprechenden Potenzmengen, zusammen. $MODEL$ kann wie folgt definiert werden:

$$\boldsymbol{MODEL} \coloneqq \underbrace{\mathbb{P}(INPUT)}_{\boldsymbol{Inputvariablen}} \times \underbrace{\mathbb{P}(STATE)}_{\boldsymbol{Zustandsvariablen}} \times \underbrace{\mathbb{P}(\widehat{STATE})}_{\boldsymbol{Startzustand}}$$

Ein Beispiel für ein solches Simulationsmodell ist *m.*

$$Sei\ m = (I, S, start)\ wobei\ I \subseteq INPUT\ , S \subseteq\ STATE\ und\ start \in \widehat{S}$$

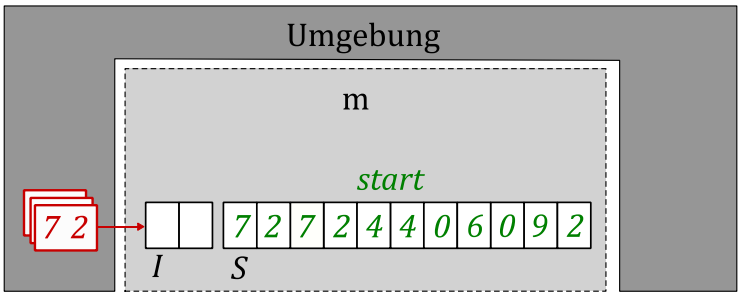



*Abbildung 7.1 Schematische Darstellung eines Simulationsmodells und seiner Umgebung.*

Abbildung 7.1 zeigt die schematische Darstellung des Modells $m$. Die Variabeln werden als Kästchen dargestellt. Neben den Zustandsvariablen $I$ und Zustandsvariablen $S$ ist auch der Startzustand $start$ (Belegungen des Zustands in Grün) für das Model $m$ dargestellt. Auch die Belegung der Eingabevariablen ist als ein Teil der Umgebung des Modells dargestellt.

Es sei an dieser Stelle angemerkt, dass sich zwei unterschiedliche Modelle keine Variablen teilen.

Sei $m_1, m_2 \in MODEL$ mit $m_1 = (I_1, S_1, start_1)$ und $m_2 = (I_2, S_2, start_2)$ dann gilt:

$$m_1 \neq m_2 \Rightarrow I_1 \cup I_2 = S_1 \cup S_2 = \emptyset$$

Wenn $m_1$ von $m_2$ verschieden sind, dann sind ihre Variablennamen disjunkt.

## 7.2 Zeit

Um das Verhalten eines Modells beschreiben zu können, ist es notwendig, zunächst einen Zeitbegriff für die betrachtete Klasse von Simulationsmodellen einzuführen.

Analog under anderem zu den Arbeiten von Broy, Lamport und Rausch (Broy, 1994; Lamport, 1989; Rausch, 2001) wird Zeit in dieser Arbeit als eine unendliche Abfolge von Intervallen

gleicher Länge verstanden. Die Länge dieses Intervalls wird als Schrittweite $\Delta t$ bezeichnet. Wir abstrahieren jedoch von dieser Länge und nutzen die natürlichen Zahlen als Zeitkoordinate.

Eine Simulation wird über einem definierten, endlichen Zeitintervall $T$ hinweg durchlaufen.

$$\boldsymbol{T} \subset \mathbb{N}^{+}$$

Ein gezeiteter Strom einer Menge $X$ ist eine Abbildung von $T$ auf Elemente dieser Menge. Entsprechend wird die Menge aller derartigen Abbildungen $X^T$ definiert.

$$X^T \coloneqq T \to X$$

Mit der Notation $x^t$ wird das Element $x$ der Menge $X$ zum Zeitpunkt $t \in T$ beschrieben.

## 7.3 Schrittfunktionen

Das Verhalten eines Simulationsmodells wird innerhalb dieser Arbeit mithilfe einer Schrittfunktion beschrieben.

Definieren wir zunächst das Universum aller möglichen Schrittfunktionen STEP als die Menge aller Abbildungen von einem Tupel aus den Belegungen einer Teilmenge der Inputvariablen und der Zustandsvariablen auf Belegungen einer Teilmenge der Zustandsvariablen.

$$STEP\colon \mathbb{P}(\widehat{INPUT}) \times \mathbb{P}(\widehat{STATE}) \to \mathbb{P}(\widehat{STATE})$$

Sei $m = (input, state, start)$ ein Modell mit den Inputvariablen $I \subseteq INPUT$ und den Zustandsvariablen $S \subseteq STATE$. Seien zudem $\hat{I}$ und $\widehat{S}$ Belegungen dieser Variaben und $start \in \widehat{S}$ ein Startzustand, dann ist die Schrittfunktion $step_m$ für dieses Modells wie folgt definiert:

$$step_m : \hat{I} \times \widehat{S} \to \widehat{S}$$

Die Funktion bildet jeweils eine Belegung der Inputvariablen und eine Belegung der Zustandsvariablen des Modells auf eine neue Belegung seiner Zustandsvariablen ab.

Sei $\mathrm{input}^{\mathrm{t}} \in \hat{I}^{\mathrm{T}}$ ein Strom der Belegungen der Eingangsvariablen von $m$.

Zusammen mit dem Startzustand $start$ kann nun mit $step_m$ das Verhalten des Modells als ein Strom von Zustandsvariablen $state^t \in \widehat{S}^T$ dargestellt werden.

$$state^0 = start \wedge \forall t \in T\colon state^{t+1} = step(input^t, state^t)$$

Diese Art der Beschreibung des Zeitverhaltens ist eine gängige Vereinfachung. Sie erlaubt die einfache Kopplung der Simulationsmodelle. Ein Beispiel für den Nutzten eines derartiges Zeitmodells ist im Kosimulations-Framework FMI (Blockwitz, 2012) zu finden. Sie entspricht den DT-Modellen.

Durch das gewählte Zeitmodell werden Probleme der Zeitsynchronisation zwischen gekoppelten Modellen, Probleme mit Sprungstellen bei kontinuierlichen Zeitmodellen und unterschiedliche Schrittweiten der Simulatoren abstrahiert. Es gibt jedoch eine Vielzahl von Ansätzen, die sich explizit mit diesen Problemen beschäftigen. Werkzeuge wie Ptolemy implementieren diese Ansätze (siehe 2.2.2). Viele von ihnen können auf den hier vorgestellten Ansatz ausgeweitet werden. Der Fokus dieser Arbeit liegt nicht auf der Zeitsynchronisation, weshalb wir im weiteren Verlauf nur noch Simulationsmodelle, deren Verhalten dem hier dargestellten entspricht, betrachten werden.

## 7.4 Abstraktion

Um später die Kopplung von Simulationsmodellen unterschiedlicher Abstraktionsebenen beschreiben zu können, müssen wir diese einer abstrakten und einer konkreten Ebene zuordnen. Dies geschieht mithilfe von zwei disjunkten Teilmengen $ABSTRACT$ und $CONCRET$.

$$ABSTRACT \cup CONCRET = MODEL$$

$$\wedge\, ABSTRACT \cap CONCRET = \emptyset$$

Hierbei fasst $ABSTRACT$ alle Modelle der abstrakten Ebene zusammen und $CONCRET$ alle Modelle der konkreten Ebene.

Da nun die Modelle ihren Ebenen zugeordnet sind, kann die Variablenabstraktion zwischen Variablen dieser Modelle beschrieben werden. Hierzu wird zunächst $LEVEL\_RELATION$, das Universum aller möglichen Relationen zwischen Belegungen der Variablen zweier Modelle definiert.

Zunächst seien ein abstraktes Modell $\overline{m}$ und ein konkretes Modell $\underline{m}$ wie folgt gegeben:

$$Seien\ \overline{m}\ \in ABSTRACT \text{ und } \underline{m} \in CONCRET$$

$$\text{wobei } \overline{m} = \left(\overline{I}, \overline{S}, \overline{s_0}\right), \underline{m} = \left(\underline{I}, \underline{S}, \underline{s_0}\right) mit\ \overline{I}, \overline{S}, \underline{I}, \underline{S} \subseteq VARIABLE\ , \overline{s_0} \in \hat{\overline{S}}\ , \underline{s_0} \in \hat{\underline{S}}$$

Zudem seien $K$ und $A$ Teilmengen der Variablen der beiden Modelle $\overline{m}$ und $\underline{m}$.

$$K \subseteq \underline{I} \cup \underline{S}, A \subseteq \overline{I} \cup \overline{S}$$

Belegungen von K werden wir später als Konkreta und Belegungen von A als Abstrakta interpretieren.

Hierfür definieren wir zunächst $LEVEL\_RELATION_{K,A}$ als das Universum aller möglichen Relationen zwischen Belegungen von $K$ und Belegungen von $A$.

$$LEVEL\_RELATION_{K,A} = \mathbb{P}(\hat{K} \times \hat{A})$$

Die Definition der Variablenabstraktion fordert, dass zwischen diesen Belegungen eine surjektive Abbildung bestehet. Deshalb werden in $ABSTRACTION\_RELATION_{K,A}$ nur die Relationen zusammengefasst, welche eine surjektive Abbildung darstellen.

$$ABSTRACTION\_RELATION_{K,A} \subset LEVEL\_RELATION_{K,A} \wedge$$

$$\forall r \in ABSTRACTION\_RELATION_{K,A}:$$

$$\forall k \in \hat{K}\ \exists!\, a \in \hat{A} : (k, a) \in r$$

Schließlich beschreibt $ABSTRACTION_{K,A}$ die Menge aller Relationen aus $ABSTRACTION\_RELATION_{K,A}$ als surjektive Funktion aufgefasst:

$$ABSTRACTION_{K,A} =$$
$$\{\, \alpha: \hat{K} \to \hat{A} \mid \exists r \in ABSTRACTION\_RELATION_{K,A} : (k, a) \in r \Leftrightarrow (\, \alpha(k) = a) \}$$

Somit beschreit $ABSTRACTION_{K,A}$ die Menge aller Variablenabstraktionen zwischen Belegungen der Variablenmenge *K* und Belegungen der Variablenmenge *A*.

Es gibt dabei keine Einschränkung bezüglich der Anzahl der Variablen in $K$ und $A$. So ist es beispielsweise auch möglich, dass zwei abstrakte mit einer konkreten Variablen in Beziehung stehen. Etwa können zwei binäre, abstrakte Variablen einer realwertigen, konkreten gegenüberstehen.

Da alle $\alpha \in ABSTRACTION_{K,A}$ surjektiv sind, besteht jedoch zwischen Belegungen von $A$ und den Belegungen von $K$ immer eine 1:n-Beziehung.

## 7.5 Integrationsoperator

Im weiteren Verlauf wird ein Integrationsoperator notwendig. Die Definition erfolgt in enger Anlehnung an den Integrationsoperator für Componentware (Rausch, 2001, S. 88), jedoch wurde die Syntax angepasst. Für die Definition wird die Projektion einer Menge von Variablenbelegungen auf eine Belegung einer Teilmenge der Variablen benötigt.

Sei $Y \subseteq X \subseteq VARIABLE$ und $x \in \hat{X}$ dann sei $x_{\restriction Y} \coloneqq \{(v \to d) \in x : v \in Y\}$ die Projektion der Belegungen von $x$ auf die Variable $Y$.

Wenn beispielsweise $X = \{a, b, c, d\}$ und $Y = \{a, b\}$ sowie $x = \{a \to 1, b \to 1, c \to 1, d \to 1\} \in \hat{X}$ eine Belegung der Variablen von X ist, dann ist $x_{\restriction Y} = \{a \to 1, b \to 1\}$. Diese Projektion wurde analog zur Formalisierung von Ptolemy (Tripakis, 2013) definiert.

Der Integrationsoperator erlaubt es nun, gezielt die Belegungen einiger Variablen einer Variablenmenge $X$ zu überschreiben. Hierfür wird die Teilmenge der Variablen von $X$, die überschrieben werden sollen, in einer Menge $Y$ zusammengefasst.

Sei $Y \subseteq X \subseteq VARIABLE\ dann$ sei

$$x \ll y \coloneqq \left(x_{\restriction (X/Y)} \cup y\right) \forall x \in \hat{X}, y \in \hat{Y}$$

Das Ergebnis der Integration $x \ll y$ ist wieder eine Belegung der Variablen aus $X$. Hierin sind alle Belegungen aus $y$ übernommen. Die Belegungen der übrigen Variablen aus $X$ (in Mengenschreibweise $X/Y$) bleiben unverändert.

Wenn beispielsweise erneut $X = \{a, b, c, d\}$ und $Y = \{a, b\}$ sowie $x = \{a \to 1, b \to 1, c \to 1, d \to 1\}$ und $y = \{a \to 0, b \to 0\}$ dann ist $(x \ll y) = \{a \to 0, b \to 0, c \to 1, d \to 1\}$.

## 7.6 Multi-Level-Modelle

In diesem Abschnitt werden Multi-Level-Modelle definiert. Sie werden genutzt, um die Struktur einer Multi-Level-Simulation zu beschreiben.

Ein Multi-Level-Modell besteht nun aus zwei Simulationsmodellen sowie zwei Tupel von Teilmengen ihrer Variablen, deren Belegungen jeweils in einer Variablenabstraktion zueinanderstehen.

$$MLMODEL \coloneqq ABSTRACT \times CONCRETE \times \left(\mathbb{P}(VARIABLE) \times \mathbb{P}(VARIABLE)\right) \times \left(\mathbb{P}(VARIABLE) \times \mathbb{P}(VARIABLE)\right)$$

Seien erneut ein abstraktes Modell $\overline{m}$ und ein konkretes Modell $\underline{m}$ wie folgt gegeben:

$$\overline{m} \in ABSTRACT \text{ und } \underline{m} \in CONCRET$$

$$\text{wobei } \overline{m} = \left(\overline{I}, \overline{S}, \overline{s_0}\right), \underline{m} = \left(\underline{I}, \underline{S}, \underline{s_0}\right) mit\ \overline{I}, \overline{S}, \underline{I}, \underline{S} \subseteq VARIABLE\ , \overline{s_0} \in \hat{\overline{S}}, \underline{s_0} \in \underline{\hat{S}}$$

Um nun definieren zu können, welche Variablen beider Modelle sich überschneiden und bei einer Kopplung synchronisiert werden müssen, werden jeweils zwei Variablenbereiche angegeben.

$$\overline{R} \subseteq \overline{I} \cup \overline{S}$$

$$\overline{W} \subseteq \overline{S}$$

$\overline{R}$ und $\overline{W}$ seien Teilmengen der Variablen des abstrakten Modells. $\overline{R}$ wird als Lesebereich des abstrakten Modells genutzt und kann sowohl Zustands- als auch Eingabevariablen enthalten. Im späteren Verlauf werden Belegungen der Variablen aus $\overline{R}$ an das konkrete Modell gesendet. $\overline{W}$ wird als Schreibbereich des Modells verwendet und enthält Zustandsvariablen. Diese Variablen werden später mit Belegungen des konkreten Modells überschrieben.

Analog seien die Lese- und der Schreibbereiche des konkreten Modells definiert.

$$\underline{W} \subseteq \underline{I} \cup \underline{S}$$

$$\underline{R} \subseteq \underline{S}$$

Die beiden Tupel $(\overline{R}, \underline{W})$ und $(\underline{R}, \overline{W})$ definieren diese Bereiche für das Multi-Level-Modell. Sie werden durch den Modellierer so gewählt, dass zwischen ihren Belegungen eine Variablenabstraktion besteht.

Somit gibt es zwei Surjektionen $\alpha_{down} \in ABSTRACTION_{\underline{W},\overline{R}}$ und $\alpha_{up} \in ABSTRACTION_{\underline{R},\overline{W}}$. Es kann jedoch nicht davon ausgegangen werden, dass diese durch den Modellierer direkt angegeben werden können. Sie werden im weiteren Verlauf lediglich als Rahmen zur Spezifikation korrekten Verhaltens einer Multi-Level-Simulation genutzt. Abbildung 7.2 stellt ein Multi-Level-Modell schematisch dar.

Auch die beiden Variablenabstraktionen $\alpha_{down}$ und $\alpha_{up}$ sind als Pfeile dargestellt. Grundsätzlich können auch Kombinationen aus einem abstrakten Modell und n konkreten Modellen auf diese Weise beschrieben werden. Hierzu muss lediglich eine Menge von Multi-Level-Modellen eingeführt werden.

$$MLMODELSET = \left\{\left(\overline{m}, \underline{m_1}, \left(\overline{R_1}, \underline{W_1}\right), \left(\underline{R_1}, \overline{W_1}\right)\right), \left(\overline{m}, \underline{m_2}, (\overline{R_2}, \underline{W_2}), \left(\underline{R_2}, \overline{W_2}\right)\right), \ldots, \left(\overline{m}, \underline{m_n}, (\overline{R_n}, \underline{W_n}), \left(\underline{R_n}, \overline{W_n}\right)\right)\right\}$$

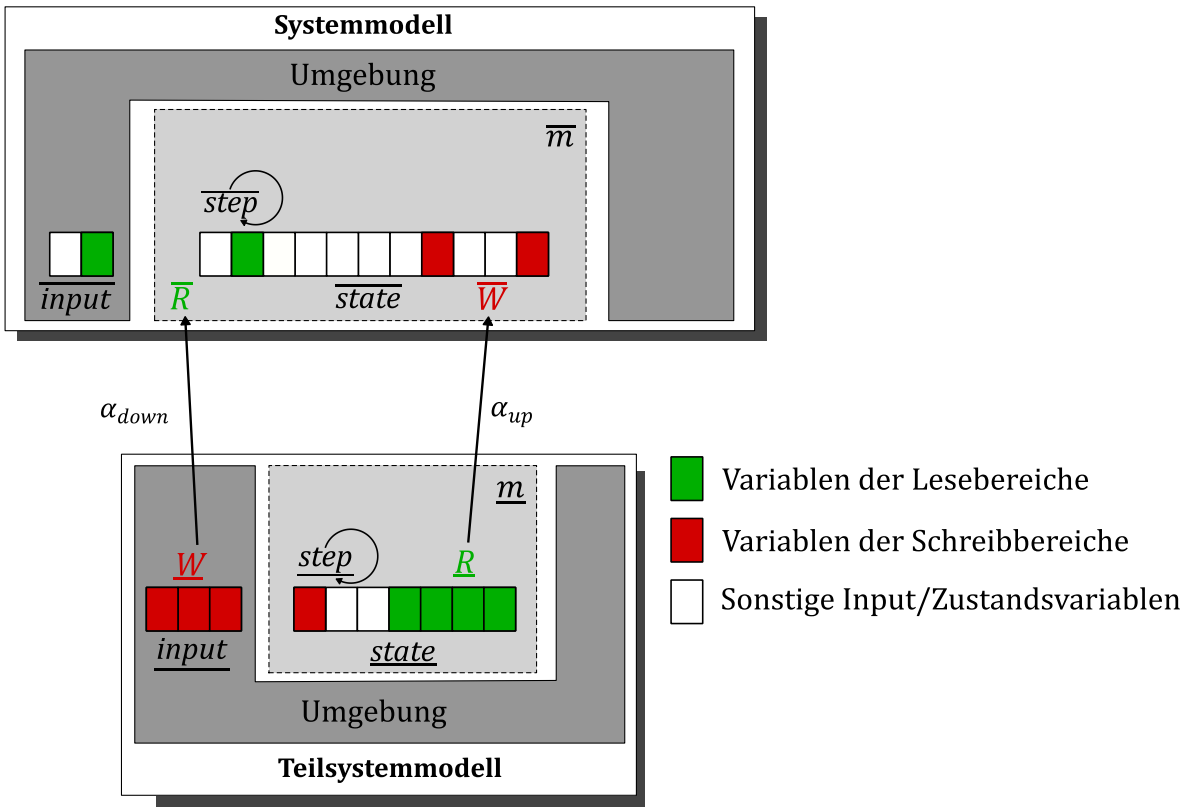



*Abbildung 7.2: Schematische Darstellung einer Multi-Level-Simulation.*

Wenn zudem die Variablenbereiche disjunkt sind, ist analog auch eine Multi-Level-Simulation möglich.

$$\forall i,j \in (1,n), i \neq j: \overline{R_i} \cup \overline{R_i} = \overline{W_i} \cup \overline{W_j} = \emptyset$$

Die Kopplung der Modelle würde unabhängig erfolgen. Allerdings werden wir uns aus Gründen der Anschaulichkeit auf den Fall nur eines konkreten Modells beschränken.

## 7.7 Multi-Level-Schritt

Auf Basis eines Multi-Level-Modells kann nun die kleinste Verhaltenseinheit einer Multi-Level-Simulation definiert werden. Sei $m_{ML} \in MLMODEL$ erneut ein Multi-Level-Modell, wie folgt definiert:

$$m_{ML} = \left(\overline{m}, \underline{m}, (\overline{R}, \underline{W}), (\underline{R}, \overline{W})\right)$$

$$\overline{m} \in ABSTRACT, \underline{m} \in CONCRETE \text{ mit } \overline{m} = \left(\overline{I}, \overline{S}, \overline{s_0}\right), \underline{m} = \left(\underline{I}, \underline{S}, \underline{s_0}\right), \overline{s_0} \in \hat{\overline{S}}, \underline{s_0} \in \hat{\underline{S}}$$

Seien zudem die *step* Funktionen beider Modelle durch $\overline{step} \in STEP_{\overline{m}}$ , $\underline{step} \in STEP_{\underline{m}}$ gegeben. Um das Verhalten eines Multi-Level-Schrittes definieren zu können, sind zudem die gezeiteten Ströme der Variablenbelegungen der Modelle erforderlich. Seien also $\overline{input^t} \in \hat{\overline{I}}^T$ und $\overline{state^t} \in \hat{\overline{S}}^T$ Belegungen der Variablen des abstrakten Modells und $\underline{input^t} \in \hat{\underline{I}}^T$ und $\underline{state^t} \in \hat{\underline{S}}^T$ Belegungen der Variablen des konkreten Modells. Beide jeweils über die gesamte Simulationszeit $\forall t \in T$.

Wir treffen die Annahme, dass vor einer Multi-Level-Simulation alle Inputs beider Ebenen bereitstehen. Diese werden während eines Multi-Level-Schrittes angepasst und an die Schrittfunktionen weitergereicht.

$$\textit{Seien } \overline{\text{input}^t} \textit{ und } \underline{\text{input}^t} \; \forall t \in T \textit{ gegeben.}$$

Nun kann ein Multi-Level-Schritt für $m_{ML}$ wie folgt definiert werden.

$$MLSTEP_{m_{ML}}: \hat{\overline{I}} \times \hat{\overline{S}} \times \hat{\underline{I}} \times \hat{\underline{S}} \rightarrow \hat{\overline{S}} \times \hat{\underline{S}}$$

Dann muss ein Multi-Level-Schritt $mls \in MLSTEP_{m_{ML}}$ die folgende Bedingung erfüllen:

$$\forall t \in T: \; mls\left(\overline{state^t}, \overline{input^t}, \underline{state^t}, \underline{input^t}\right) = \begin{pmatrix} \overline{state^{t+1}} \\ \underline{state^{t+1}} \end{pmatrix} \Rightarrow$$

$$\overline{state^0} = \overline{s_0} \wedge \underline{state^0} = \underline{s_0} \wedge$$

$$\forall t \in (T \backslash 0) \; \exists \, y \in \widehat{\underline{W}} :$$

$$\alpha_{down}(y) = \left(\overline{state^t} \cup \overline{input^t}\right)_{\restriction \overline{R}} \wedge$$

$$\underline{input}^{t^*} = \underline{input^t} \ll y \wedge$$

$$\underline{state}^{t^*} = \underline{state^t} \ll y \wedge$$

$$\underline{state^{t+1}} = \underline{step}\left(\underline{input}^{t^*}, \underline{state}^{t^*}\right) \wedge$$

$$\overline{state^{t+1}}^* = \overline{step}\left(\overline{input^t}, \overline{state^t}\right)$$

$$\overline{state^{t+1}} = \overline{state^{t+1}}^* \ll \alpha_{up}\left(\left(\underline{state^{t+1}}\right)_{\restriction \underline{R}}\right)$$

Intuitiv kann diese Formel wie folgt aufgefasst werden.

Wenn ein Multi-Level-Schritt $mls$ die Belegung der Zustands- und Eingangsvariablen $\overline{state^t}, \overline{input^t}$ eines abstrakten Modells, sowie die Belegung der Zustands- und Eingangsvariablen $\underline{state^t}, \underline{input^t}$ eines konkreten Modells, auf jeweils eine Folgebelegung $\overline{state^{t+1}}$ bzw. $\underline{state^{t+1}}$ der Zustandsvariablen beider Modelle abbildet, dann müssen die folgenden Bedingungen gelten (Implikationspfeil in der Formel).

Die Belegungen zum Zeitpunkt *t=0* der Zustandsvariablen beider Modelle $\overline{state^0}$ und $\underline{state^0}$sind jeweils durch den Startzustand der Modelle $\overline{s_0}$ und $\underline{s_0}$ gegeben.

Für alle anderen Zeitpunkte muss es eine Belegung $y$ des Schreibbereiches $\underline{W}$ des konkreten Modells geben, die zur einer Belegung $\left(\overline{state^t} \cup \overline{input^t}\right)_{\restriction \overline{R}}$ des Lesebereiches für den Zeitpunkt abstrahiert werden kann: $\alpha_{down}(y) = \left(\overline{state^t} \cup \overline{input^t}\right)_{\restriction \overline{R}}$

Da viele $y$ existieren, welche dieser Bedingung gerecht werden, wird hiermit die Äquivalenzrelation der Variablenabstraktion modelliert. Hierdurch wird die Konsistenzanforderung (vgl. 4.3.1) erfüllt.

Die Zwischenbelegungen $\underline{input^{t^*}}$ und $\underline{state^{t^*}}$ sind die Integration von diesem $y$ in die aktuellen Belegungen $\underline{input^t}$ und $\underline{state^t}$ der Variablen des konkreten Modells.

Der Folgezustand ist ein Schritt des konkreten Modells mit diesen Zwischenbelegungen $\underline{state^{t+1}} = \underline{step}\left(\underline{input^{t^*}}, \underline{state^{t^*}}\right)$.

Der Zwischenzustand des abstrakten Modells ist ein Schritt des abstrakten Modells mit den aktuellen Belegungen seiner Variablen $\overline{state^{t+1}}^* = \overline{step}\left(\overline{input^t}, \overline{state^t}\right)$.

Der Folgezustand $\overline{state^{t+1}}$ des abstrakten Modells ist die Integration der Abstraktion des Lesebereiches des neuen Zustandes des detaillierten Modells in den Zwischenzustand $\overline{state^{t+1}} = \overline{state^{t+1}}^* \ll \alpha_{up}\left(\left(\underline{state^{t+1}}\right)_{\restriction \underline{R}}\right)$.

## 7.8 Berechnungsvorschrift eines Multi-Level-Schritts

In diesem Abschnitt betrachten wir nun eine intuitive Berechnungsvorschrift für $mls \in MLSTEP_{m_{ML}}$. Diese ist in Abbildung 7.3 dargestellt. Sie lässt sich grob in die Phasen *Verteilen*, *Paralleles Verarbeiten* und *Integrieren* unterteilen.

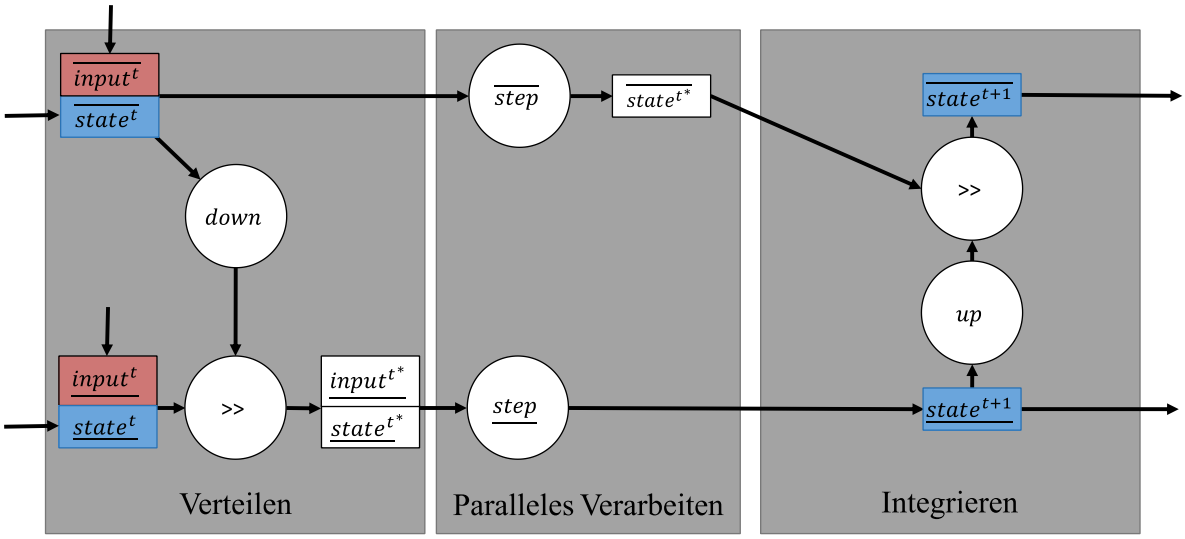



*Abbildung 7.3: Intuitive Berechnungsvorschrift für einen Multi-Level-Schritt.*

Um von einem Zustand zum nächsten zu gelangen, muss zunächst der Zustand des abstrakten Modells verteilt werden. Dieses erfolgt, indem eine Belegung für die Variablen $\underline{W}$ ermittelt wird, welche der Belegung von $\overline{R}$ aus dem aktuellen Zeitschritt entspricht. In der formalen Spezifikation wurde diese Belegung als $y$ bezeichnet. Da es mehr als eine Belegung von $\underline{W}$ gibt, die zu $\overline{R}$ abstrahiert werden kann, wird der Algorithmus down eingeführt. Dieser sucht eine solche Belegung aus. Anschließend wird die gefundene Belegung von $\underline{W}$ in $\underline{\text{input}^t}$ und $\underline{state^t}$ integriert. Es entsteht ein Zwischenschritt, der in der Abbildung als $\underline{input^{t^*}}$ und $\underline{state^{t^*}}$ dargestellt ist.

In der Phase *Paralleles Verarbeiten*, werden nun - wie im nicht gekoppelten Fall - die Schrittfunktionen $\overline{step}$ und $\underline{step}$ angewandt. Auf der abstrakten Ebene entsteht so der Zwischenzustand $\overline{state^t}^*$.

In der Phase *Integration* wird nun mithilfe eines Algorithmus up der neue Zustand der konkreten Simulation $\underline{state^{t+1}}$ auf die abstrakte Ebene abgebildet und in den Zwischenzustand $\overline{state^t}^*$ integriert. Hierdurch entsteht $\overline{state^{t+1}}$, der neue Zustand des abstrakten Modells.

Die intuitive Berechnungsvorschrift folgt dieser Spezifikation, sofern die Algorithmen $down$ und $up$ wiederum bestimmte Anforderungen erfüllen. Diese Anforderungen werden in den Abschnitten 7.9 und 7.10 dargestellt.

## 7.9 Anforderungen an den Algorithmus *up*

Der Algorithmus $up$ wandelt eine Belegung der Variablen des Lesebereiches $\underline{R}$ des konkreten Modells in eine Belegung der Variablen des Schreibbereiches $\overline{W}$ des abstrakten Modells um.

**Der Algorithmus $up$ muss die Variablenabstraktion nachbilden.** Die Ausgabe von $up$ zu einem gegebenen Konkretum $k \in \hat{\underline{R}}$ muss genau jenes Abstraktum $a \in \widehat{\overline{W}}$ sein, welches diesem durch die Variablenabstraktion zugeordnet wird.

$$up(k) = \alpha_{up}(k) \forall\, k \in K$$

**Der Algorithmus $up$ muss determiniert sein**, da eine Variablenabstraktion $\alpha$ dem Konkretum $k$ nur ein einziges solches Abstraktum zuordnet.

## 7.10 Anforderungen an den Algorithmus *down*

Der Algorithmus down wandelt eine Belegung der Variablen des Lesebereiches im abstrakten Modell (z.B. das Volumen eines abstrakten Paketes) in eine Belegung des Schreibbereiches des konkreten Modells um (z.B. die Dimensionen Höhe, Breite und Tiefe eines detaillierten Paketes).

Für ein gegebenes Abstraktum $a \in \widehat{\overline{R}}$ gibt es jedoch eine ganze **Äquivalenzklasse** $[k]_{\sim} = \left\{ k \in \underline{\widehat{W}} \middle| \alpha_{down}(k) = a \right\}$ zugehöriger Konkreta.

Aus diesem Grund wird ein Algorithmus $down$ benötigt, der bei jedem Aufruf nur ein einziges solches Konkretum aus der Äquivalenzklasse selektiert.

**Der Algorithmus $down$ kann nicht-determiniert sein.** Ein determinierter Algorithmus wird bei jeder Umwandung des Abstraktums $a$ dasselbe Konkretum $k$ ausgeben. Damit wird immer nur ein einziger Repräsentant jeder Äquivalenzklasse $[k]_{\sim}$ als Eingabe genutzt. Dies würde dazu führen, dass das Teilsystem nur mit einem stark eingeschränkten Unterraum der möglichen Variablenbelegungen simuliert wird. Alle Konkreta einer gegebenen Äquivalenzklasse, bis auf eines, würden ignoriert, ungeachtet ihrer möglichen Relevanz für die Domäne. Ein Beispiel für ein determiniertes $down$ ist die Funktion f aus Abschnitt 3.10.4. Sie bildet nur auf den Unterraum aller kubischen Pakete ab. Das heißt, dass zwei Aufrufe von $down$ mit demselben Input unterschiedliche Ausgaben produzieren können.

**Der Algorithmus $down$ muss der Verteilungsanforderung entsprechen.** Wie bereits in 4.3.2 diskutiert, müssen die Konkreta, die innerhalb der Multi-Level-Simulation erzeugt werden, der bedingten Wahrscheinlichkeitsverteilung $P(M^* = k | M = a)$ folgen. Diese gibt der

Modellierer indirekt durch das Angeben von Inputs für die isolierte Detailsimulation an. Somit gilt für $down$ die folgende Anforderung:

$$\forall a \in \widehat{\widehat{R}}, k \in \underline{\widehat{W}} : P(down(a) = k) = P(M^* = k | M = a)$$

Die Wahrscheinlichkeit, dass $down$ einem gegebenen $a$ ein bestimmtes $k$ zuordnet, entspricht der Wahrscheinlichkeit für das entsprechende Ereignis in der wahren Verteilung.

Wie bereits dargestellt, weist P inkonsistenten Paaren $(a, k)$ die Wahrscheinlichkeit 0 zu.

$$P(\{(a, k) : a \neq \alpha(k)\}) = 0$$

Deshalb beinhaltet diese Anforderung an die Verteilung auch die Konsistenzanforderung für den Bild- und Wertebereich von $down$.

$$down(a) = k \;\Rightarrow\; \alpha_{down}(k) = a$$

Durch diese Verteilungsanforderung wird der Algorithmus $down$ nur Konkreta ausgeben, die zu dem eingegebenen Abstraktum passen.

# 8 Lernende Multi-Level-Simulation

Ein zentraler Beitrag dieser Arbeit liegt in dem Ansatz einer lernenden Multi-Level-Simulation. Dieses Lösungskonzept stellt die Antwort auf die in Abschnitt 6 aufgeworfene Forschungsfrage dar. Die grundlegende Idee der lernenden Multi-Level-Simulation besteht darin, $up$ und $down$ nicht explizit zu beschreiben, sonden anhand von Beispielen zu lernen. Die zugrunde liegende Struktur einer Multi-Level-Simulation wurde bereits in den Abschnitten 7.1 bis 7.7 definiert. Ebenso wurde in Abschnitt 7.8 ein Berechnungsvorschrift zur Ausführung einer Multi-Level-Simulation beschrieben, wobei $up$ und $down$ als gegeben angesehen wurden.

Dieser Abschnitt konzentriert sich auf den Ablauf, welcher den Rahmen für das Lernen von $up$ und $down$ darstellt. Abbildung 8.1 (auf der nächsten Seite) zeigt diesen Ablauf. Die Grafik hat dabei einen zusammenfassenden Charakter und greift bereits Notationen der kommenden Kapitel auf. Die Abbildung wird durch das Systemmodell (oben) und das Teilsystemmodell (unten) eingerahmt. Die Algorithmen $up$ und $down$ verbinden die beiden Modelle als Pfeile. Im Zentrum finden sich die Abläufe $up$ und $down$ zu bestimmen.

Hierbei baut die Bestimmung von $down$ auf der Bestimmung von $up$ auf. Zunächst wird eine kleine Anzahl von manuell gelabelten Trainingsdaten zur Bestimmung von $up$ genutzt (in der Abbildung 1.). Dies wird in Abschnitt 8.1 dieses Kapitels dargestellt. Anschließend wird $up$ zur Generierung eines großen Trainingssets für die Bestimmung von $down$ verwendet (in der Abbildung 2.). Dies wird in Abschnitt 8.2 beschrieben.

Sowohl bei der Bestimmung von $up$ als auch bei der Bestimmung von $down$ werden Fehler eingeführt. Da $up$ über das automatische Labeling einen Einfluss auf $down$ hat, muss der Fehler, den $up$ einbringt ($MAE$), und der Fehler, den $down$ beiträgt ($ARTE$), definiert und in Zusammenhang gebracht werden (Abschnitt 8.3). Die Fehler finden sich als Kästen oben rechts an 1. und 2.

Zur Vereinfachung der Darstellung wird im weiteren Verlauf $\alpha_{down} \equiv \alpha_{up}$ verwendet. Dies entspricht dem Beispiel aus Abschnitt 3, in dem $\underline{W}$ sowie $\underline{R}$ konkreten Paketen und $\overline{W}$ sowie $\overline{R}$ abstrakten Paketen entsprechen. Somit sind die Variablenabstraktionen ($\alpha_{down} \in ABSTRACTION_{\underline{W},\overline{R}}$ und $\alpha_{up} \in ABSTRACTION_{\underline{R},\overline{W}}$) identisch ($\alpha_{down} \equiv \alpha_{up}$). Hierdurch ergibt sich die Verbindung zwischen dem Bestimmen von $up$ und $down$ wie sie in Abbildung 8.1 dargestellt ist. Der Definitionsbereich von $up$ ($\underline{R}$) entspricht dem Wertebereich von $down$ ($\underline{W}$) und umgekehrt.

Dies stellt jedoch keine prinzipielle Einschränkung des Lösungsansatzes dar. Unterscheiden sich $\alpha_{down}$ und $\alpha_{up}$, können beide Variablenabstraktionen separat behandelt werden. Für $\alpha_{up}$ wird erneut lediglich $up$ bestimmt. Für $\alpha_{down}$ wird zunächst ein temporäres $up^*$, welches den

Wertebereich $\underline{W}$ auf den Bildbereich $\overline{R}$ abbildet, ermittelt. Anschließend kann $up^*$ für das automatische Labeln genutzt werden, um Trainingsdaten zur Bestimmung von $down$ zu generieren.

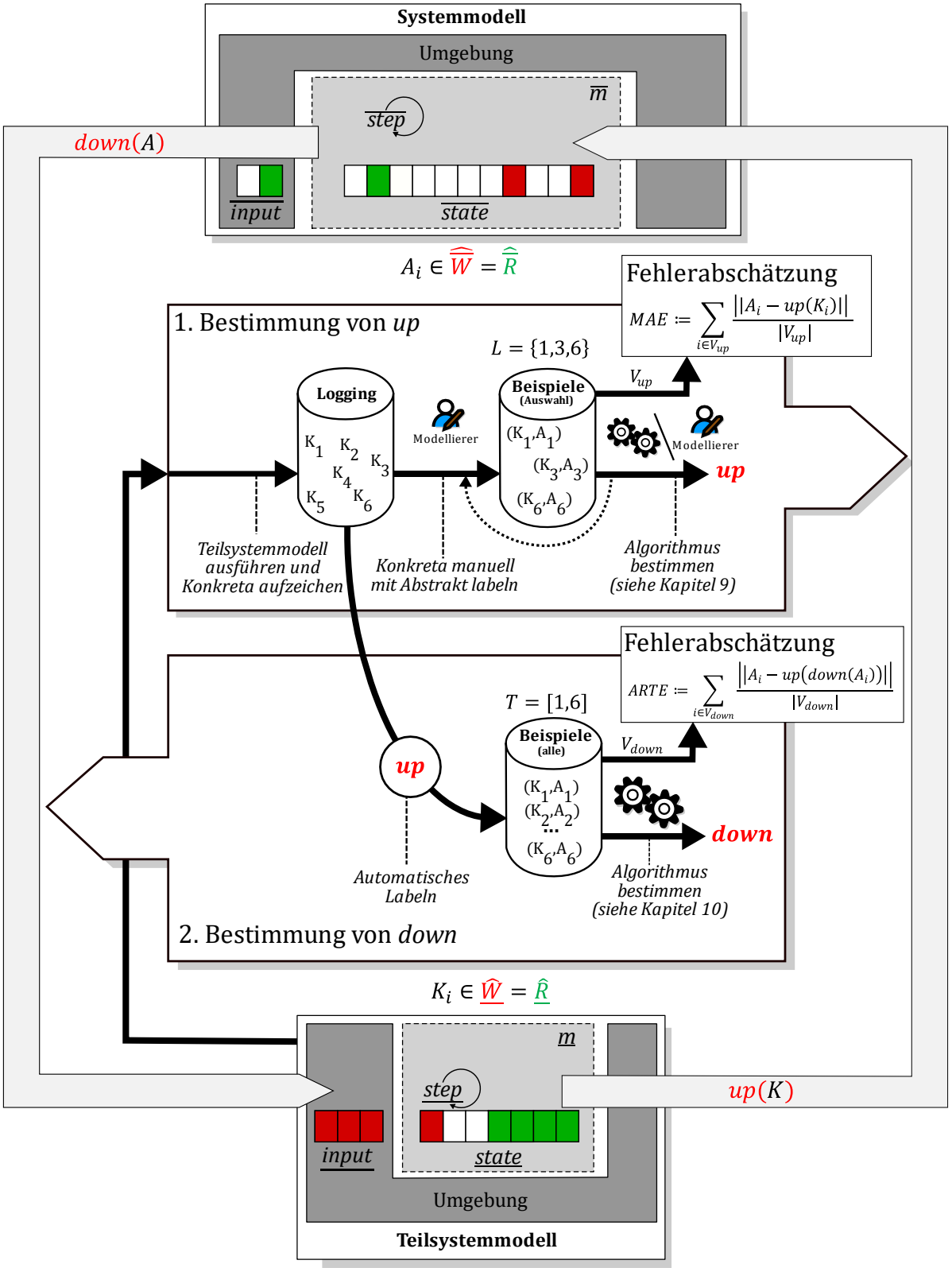



*Abbildung 8.1: Ablauf für das Lernen von up und down.*

## 8.1 Bestimmung von *up*

**Historie der Konkreta.** Die Basis zur Bestimmung beider Algorithmen liegt in der Aufzeichnung von Konkreta. Hierzu wird das untersuchte Teilsystemmodell isoliert ausgeführt. Die Belegungen aller Variablen des Lese- und des Schreibbereiches werden gespeichert. Der Modellierer selektiert, welche Variablen als Konkreta zusammengefasst werden. Durch diese Auswahl bestimmt er indirekt den Wertebereich von $up$ und den Bildbereich von $down$. Falls der Schreibbereich eine Teilmenge der Input-Variablen ist ($\underline{W} \subseteq \underline{input}$), kann auf das Ausführen des Teilsystemmodells verzichtet werden. Stattdessen können die Konkreta direkt aus der Inputsequenz abgeleitet werden. Das Ergebnis dieses Schrittes ist eine Menge von Belegungen der konkreten Variablen - die Konkreta $K_1$ bis $K_n$.

**Manuelles Labeln.** Die Historie enthält nun eine Menge von Beispielen für Konkreta. Um diese Menge als Trainingsdaten für $up$ nutzen zu können, sind jedoch noch Labels notwendig. Diese werden für eine kleine Anzahl durch einen Modellierer manuell definiert. Der Modellierer entscheidet selbst, welche Konkreta er labelt. Er mach dies auf Basis seiner Erfahrung in der Domäne. Das Ergebnis dieses Schritts ist eine kleine Menge von gelabelten Trainingsdaten.

**Lernen von $up$.** Die eigentliche Bestimmung des Algorithmus $up$ kann als ein Regressionsproblem formuliert werden. Gesucht ist eine Funktion zwischen dem Wertebereich der Konkreta und dem Bildbereich der Abstrakta, welche die Trainingsdaten mit einem möglichst geringen Fehler abbildet. In Kapitel 10 wird die Auswahl eines geeigneten Verfahrens zur Lösung dieses Regressionsproblems thematisiert. Hierbei ist zu berücksichtigen, dass nur wenige, dafür explizit ausgewählte, Beispiele als Trainingsmenge zur Verfügung stehen. Das Ergebnis dieses Schrittes ist eine ausführbare Beschreibung von $up$.

## 8.2 Bestimmung von *down*

**Automatisches Labeln.** Mit Hilfe der ausführbaren Beschreibung von $up$ kann nun die gesamte Historie der Konkreta mit Labeln versehen werden. Durch dieses Vorgehen wird der Umfang dieser Beispielmenge lediglich durch die Anzahl der für das Teilsystemmodell vorhandenen Inputbelegungen beschränkt.

**Lernen von $down$.** Bei der Bestimmung von $down$ müssen die in Abschnitt 7.10 formulierten Anforderungen eingehalten werden. So darf $down(a)$ nur Konkreta $y$ ausgeben, die tatsächlich in der Äquivalenzklasse von $a$ liegen. Zudem muss die Verteilung der nicht-determiniert erzeugten Konkreta korrekt sein.

In dieser Arbeit werden diese Aspekte - Konsistenz und der Verteilungskonformität von $down$ - dadurch zusammengeführt, dass wir die bedingte Wahrscheinlichkeit $P(M^* = k|M = a)$ als statistisches Modell für $down$ zugrunde legen. In dieser Arbeit wird vorgeschlagen, $down$ als Zufallsgenerator zu modellieren, dessen Ausgaben unter Berücksichtigung der Eingabe $a$, eben dieser Wahrscheinlichkeitsverteilung folgt. Es wird nicht gefordert, dass der Modellierer diese Verteilung explizit angibt. Vielmehr wird ausgenutzt, dass diese Verteilung in gelabelten Beispielen, welche im letzten Schritt in großer Zahl produziert wurden, bereits enthalten ist. Abschnitt 10 zeigt auf, wie es möglich ist, aus diesen Beispielen ein geeignetes $down$ zu ermitteln. Das Ergebnis dieses Schritts ist eine ausführbare Beschreibung von $down$.

## 8.3 Fehlerabschätzung

In diesen Abschnitt werden die Fehlerabschätzungen $MEA$ und $ARTE$ eingeführt. Sie erlauben es, die Genauigkeit von $up$ und $down$ zu untersuchen. Dies ist die Basis für die Auswahl geeigneter Verfahren.

**MEA.** In Abschnitt 7.9 wurde gefordert, dass $up$ die Variablenabstraktion nachbildet.

$$up(k) = \alpha_{up}(k) \forall\, k \in K$$

Da die Variablenabstraktion nicht bekannt ist, ist es zweckmäßig, den *mean absolut error* ($MAE$) bezüglich einer Validierungsmengen als Fehlermaß zu verwenden. Um dieses Maß zu bestimmen, werden die manuell erzeugten Paare $(K_i, A_i)\ mit\ i \in L_{up} \subseteq \mathbb{N}$ zunächst in disjunkte Trainings- und Validierungsmengen ($T_{up}$ und $V_{up}$) aufgeteilt. Seien $T_{up}, V_{up} \subset L_{up}$ mit

$$T_{up} \cup V_{up} = L_{up} \wedge\ T_{up} \cap V_{up} = \emptyset$$

Nun wird $up$ ausschließlich auf Basis von $T_{up}$ bestimmt. Anschließend kann $MAE$ mithilfe von $V_{up}$ wie folgt definiert werden:

$$MAE := \sum_{i \in V_{up}} \frac{\big||A_i - up(K_i)|\big|}{|V_{up}|}$$

Hierbei ist $\big||.|\big|$ die euklidische Norm und $|.|$ die Mächtigkeit einer Menge.

**ARTE.** In Abschnitt 7.10 fordern wir von $down$, zu einem gegebenen Abstraktum $a$ nur Konkreta zu produzieren, die auch zu diesem Abstraktum passen.

$$down(a) = k \ \Rightarrow\ \alpha_{down}(k) = a$$

Hieraus wird im Rahmen der vorliegenden Arbeit eine Fehlerabschätzung abgeleitet. Der *absolute round trip error* ($ARTE$).

Durch das automatische Labeling mit $up$ entsteht eine Menge von Paaren $(K_j, A_j) mit\ j \in L_{down}$. Um $ARTE$ bestimmen zu können, wird auch diese Menge in disjunkte Teilmengen aufgeteilt. Seinen $T_{down}, V_{down} \subset L_{down}$ mit

$$T_{down} \cup V_{down} = L_{down}, wobei\ T_{down} \cap V_{down} = \emptyset$$

Die Menge $T_{down}$ ist nun die Grundlage für das Training von $down$. Auf Basis der Menge $V_{down}$ lässt sich nun $ARTE$ wie folgt definieren:

$$ARTE := \sum_{i \in V_{down}} \frac{\left|\left|A_i - up(down(A_i))\right|\right|}{|V_{down}|}$$

Hierbei ist zu beachten, dass der $ARTE$ relativ zu dem, ebenfalls fehlerbehafteten, $up$ definiert ist. Dies erlaubt es, trotz des verketteten Vorgehens, die Fehler isoliert zu betrachten. Die Summe beide Fehler liefert einen Worst-Case für den Gesamtfehler.

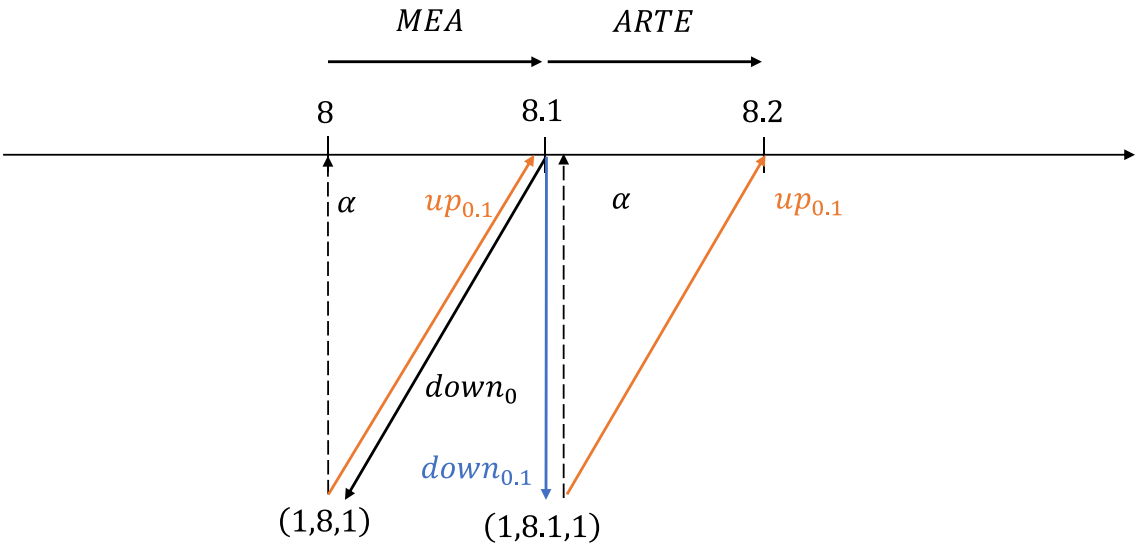



*Abbildung 8.2: Beispiel für das Zusammenspiel von MEA und ARTE.*

Abbildung 8.2 illustriert die beiden Fehler und deren Zusammenhang anhand des Lieferkettenbeispiels aus Abschnitt 3. Die Abstraktionsfunktion $\alpha(h, b, t) = h * b * t$ weist dem Paket (1,8,1) das Volumen 8 zu. Diese Funktion muss als unbekannt angenommen werden.

Definierten wir nun ein fehlerbehaftetes $up_{0.1}$ wie folgt:

$$up_{0.1}(h, b, t) = (h * b * t) + 0.1$$

Der Bias führt zu einem $MAE$ von 0.1. In Abbildung 8.2 bildet $up_{0.1}$ das Paket (1,8,1) auf 8.1 ab, was einem Fehler von 0.1 entspricht. Der Index weist auf diesen $MAE$ hin.

Ein perfektes $down_0$, mit $ARTE = 0$, gibt für die Eingabe 8.1 nun ausschließlich Pakete aus, denen $up_{0.1}$ ebenfalls das Volumen 8.1 zuweist. Dies ist jedoch nicht die Äquivalenzklasse bezüglich $\alpha$, also $[(1,8,1)]_{\sim}$, sondern eine neue Äquivalenzklasse $[(1,8,1)]_{up_{0.1}}$, die durch das fehlerbehaftete $up_{0.1}$ induziert wird. Alle Pakete in $[(1,8,1)]_{up_{0.1}}$ werden von $up_{0.1}$ auf 8.1 abgebildet und weichen dabei im Durchschnitt um 0.1 von dem Volumen ab, welches $\alpha$ zuweist.

Sei nun $down_{0.1}$ mit einem $ARTE$ von 0.1 behaftet. Wenn wir dieses $down_{0.1}$ auf 8.1 anwenden, erhalten wir beispielsweise das Paket (1, 8.1, 1). Dieses ist bezüglich $\alpha$ sogar korrekt. Das fehlerbehaftete $up_{0.1}$ weist diesem Paket jedoch das Volumen 8.2 zu. Dies weicht um 0.1 von dem Eingabewert 8.1 ab und entspricht somit dem *ARTE* von 0.1.

Der durchschnittliche Fehler relativ zu $\alpha$ wird die Summe beider Fehler nicht überschreiten. Bei höherdimensionalen Abstrakta wird es unwahrscheinlicher, dass sich der Fehler aufhebt, allerdings bleibt die obere Schranke aus $MAE + ARTE$ bestehen.

# 9 Maschinelles Lernen zur Bestimmung von *up*

Dieses Kapitel beschreibt, wie es möglich ist, Verfahren des maschinellen Lernens zu nutzen, um den Algorithmus *up* anhand kleiner Mengen gelabelter Beispiele zu lernen.

Entsprechend seiner Anforderungen (siehe Abschnitt 7.9) muss *up* deterministisch die unbekannte Variablenabstraktion $\alpha_{up}$ nachbilden. Da $\alpha_{up}$ definitionsgemäß (siehe Abschnitt 7.4) eine surjektive Abbildung ist, handelt es sich bei der Bestimmung von *up* um ein Regressionsproblem.

Zur Lösung derartiger Probleme gibt es eine Reihe von etablierten Verfahren. Die ausgewählter Verfahren zur Bestimmung von *up* wurde bereits betrachtet (Wittek, 2018). Hierzu wurden unter Anderem vier Trainingsmengen mit jeweils 20 Datenpunkten ausgewählt. Als Abstraktionsfunktion wurden dabei Summe, Mittelwert und Median sowie die Projektion auf eine Komponente genutzt. Ausgewertet wurde der *mean squared error*. Die besten Ergebnisse aller Verfahren waren für:

- die Projektion MSE=0,0015 (MEA=0,039),
- die Summe MSE=5,25E-9 (MEA=7,25E-5),
- den Mittelwert MSE=4,88E-10 (MEA=2,21E-5) und für
- den Median MSE=0,0126 (MEA=0,11).

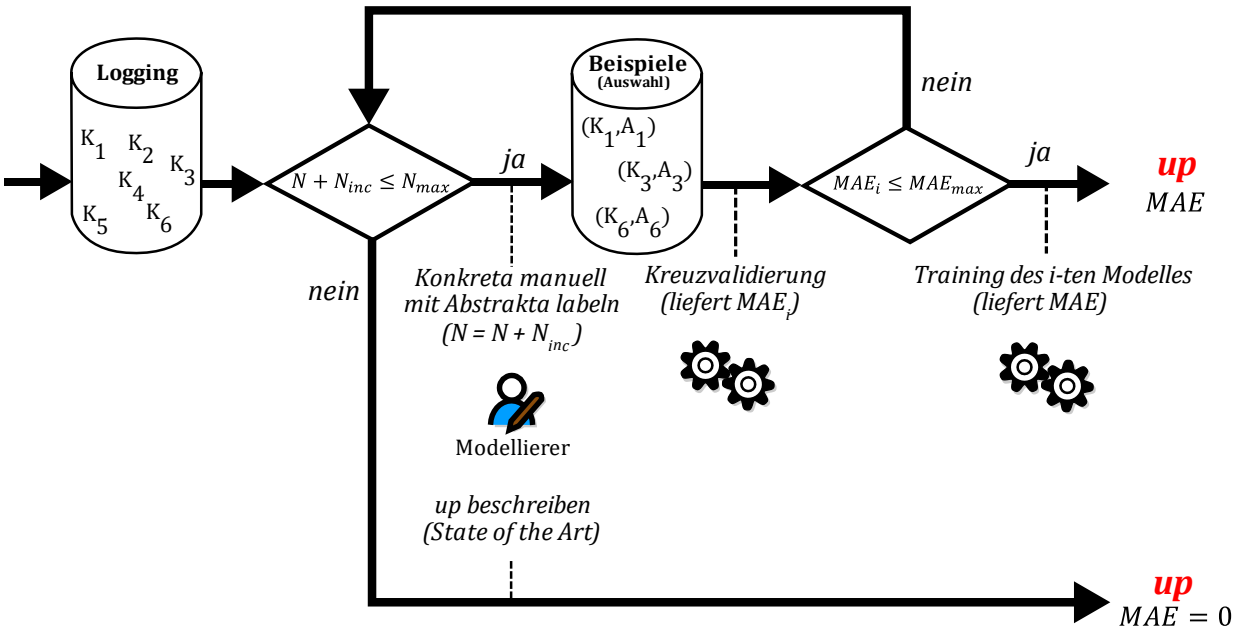



*Abbildung 9.1: Prozess zur Bestimmung von up.*

Allerdings erzielten diese Ergebnisse nicht ein einzelnes Verfahren. Auch war keines der Verfahren dominant gegenüber einem der anderen Verfahren. Zudem schwankte der erzielbare MAE zwischen den Beispielen.

Aus diesem Grund wird das Bestimmen von $up$ im Rahmen dieser Arbeit in einen einfachen Prozess zerlegt. Abbildung 9.1 gibt einen Überblick über diesen Prozess.

Zunächst ist der Modellierer aufgefordert, eine Grenze für den maximal zulässigen Fehler $MAE_{max}$ und eine Grenze für die maximale Zahl der Beispiele $N_{max}$, die er zu labeln bereit ist, anzugeben. Als nächstes labelt der Modellierer die ersten $N = 0 + N_{inc}$ Beispiele, die nun als Trainingsdaten genutzt werden.

Es wird nach einem Modell gesucht, welches den Fehler auf Basis der vorhandenen $N$ Beispielen unterschreitet. Dabei wird auch eine Veränderung der Parameter eines Modells, wie etwa eine veränderte Anzahl der Layer eines neuronalen Netzes, als ein eigenes Modell aufgefasst.

Zur Bewertung der Modelle wird Kreuzvalidierung eingesetzt. Dabei werden die Trainingsdaten in $k$ gleichgroße Teilmengen aufgeteilt. Jedes der Modelle wird nun $k$-mal trainiert. Dabei wird jeweils eine der $k$ Teilmengen nicht zum Training verwendet, sondern zur Ermittlung eines Fehlermaßes (in unserem Fall dem *MAE*). Dem $i$-ten Modell wird dann der durchschnittliche Fehler über alle $k$ Durchläufe zugewiesen (vgl. (Bishop, 2006, S. 33)). In unserem Fall wird dieser durchschnittliche Fehler mit $MAE_i$ bezeichnet.

Wenn eines der Modelle den geforderten Fehler unterschreitet, wird in einem zweiten Schritt das Modell mit dem kleinsten Fehler ausgewählt und erneut trainiert, diesmal mit allen Trainingsdaten. Zudem wird mit dem trainierten $up$ der MAE mit einer weiteren, separaten Validierungsmenge bestimmt.

Unterschreitet keines der Modelle den geforderten $MAE_{max}$, gibt der Modellierer $N_{inc} \in \mathbb{N}$ weitere Beispiele an und das Verfahren wird mit den nun $N = N + N_{inc}$ Beispielen erneut durchlaufen.

Dies wird wiederholt, bis ein geeignetes Modell gefunden ist, oder bis der Modellierer nicht mehr bereit ist, weitere Beispiele anzugeben ($N + N_{inc} > N_{max}$). In diesem Fall muss der Modellierer schließlich die $up$ Funktionen manuell beschreiben, was dem Stand der Forschung entspricht.

Im weiteren Verlauf dieses Kapitels wird untersucht, ob es prinzipiell möglich ist, Abstraktionsfunktionen mithilfe des dargestellten Prozesses automatisiert zu bestimmen.

Hierbei wird ausschließlich auf wohlbekannte Modelle und Trainingsverfahren zurückgegriffen.

In Abschnitt 9.1 werden aus bestehenden Arbeiten zur abstraktionsübergreifenden Simulation insgesamt 6 Trainingsmengen abgeleitet und eine Modellauswahl entsprechend des vorgeschlagenen Prozesses durchgeführt. Abschließend werden die Ergebnisse in Abschnitt 9.2 zusammengefasst und in Abschnitt 9.3 diskutiert.

## 9.1 Benchmark

In diesem Abschnitt wird der Ansatz, $up$ mithilfe von wohlbekannten Regressionsverfahren zu lernen, untersucht. Zunächst werden dazu in Abschnitt 9.1.1 aus den in Kapitel 5 beschriebenen Arbeiten Datensätze abgeleitet. In Abschnitt 9.1.2 werden zudem die verwendeten Regressionsmodelle und deren Parameter dargestellt.

### 9.1.1 Datensätze

Wie bereits in Kapitel 5 dargestellt, gibt es eine Reihe von Arbeiten, welche die Kosimulation auf heterogenen Abstraktionsstufen thematisieren. Auch, wenn sich diese Ansätze in Hinblick auf die Anforderungen einer Multi-Level-Simulation als ungeeignet herausgestellt haben, enthalten sie dennoch relevante Beispiele für Abstraktionsfunktionen. In der Regel werden diese Funktionen als Teil eines Proof of Concepts der Ansätzen explizit angegeben und stehen somit für den Benchmark zur Verfügung.

Die folgenden Abstraktionsfunktionen konnten auf diese Weise extrahiert werden:

**Summe (SUM)**. Die Arbeit von Navarro (Navarro, 2011) befasst sich mit der agentenbasierten Simulation von Städten. Dabei können die Agenten über Ressourcen, wie zum Beispiel Bargeld, verfügen. Wenn einzelne Agenten zu einer Gruppe abstrahiert werden, müssen die Ressourcen der Gruppe summiert werden. Die Arbeit von Huber (Huber, 2011) befasst sich mit diskreten Materialflusssimulationen. Dabei werden Ausfallzeiten einer Gruppe von seriellen Maschinen summiert. Auch die Abbildung eines Energieflusses aus separaten Phasen zu einem einzigen Energiefluss bei Schütte (Schütte, 2013, S. 53) nutzt die Summenfunktion.

$$SUM: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$SUM(X) = \sum_{i=1}^{n} X_i$$

**Durchschnitt eines Vektors (MEAN)**. Bei Huber (Huber, 2011) wird die Auslastung von Maschinen bei deren Abstraktion zu Maschinengruppen als Durchschnitt modelliert. Bei

Navarro (Navarro, 2011) werden physische und psychische Eigenschaften der Agenten durch die Durchschnittbildung zu einer Gruppe abstrahiert. Bei Rao (Rao, 2000) wird der Durchschnitt in einer Reihe von Beispielen verwendet, um zu motivieren, dass bei der Abstraktion grundsätzlich ein Informationsverlust stattfindet. In der Arbeit von Hong (Hong, 2013), welche sich mit der militärischen Luftraumüberwachung befasst, wird der Schaden, den ein Kampfflugzeug erlitten hat, in einen Gesundheitszustand des Piloten und den Schadenszustand des Flugzeugmotors aufgeschlüsselt. Dies wird stellvertretend für den Schadenszustand aller Subsysteme des Flugzeuges beschrieben. Dieses Schadensbild wird bei der Abstraktion zu einem Flugzeug mit einem einzigen Schadenwerts als Durchschnitt abgebildet. Diese Beispiele werden erweitert, indem nicht nur ein Durchschnitt, sondern ein gewichteter Durchschnitt mit den Gewichten $g_i$ verwenden wird. Die beiden Beispiele können dadurch modelliert werden, dass die Gewichte gleich gewählt werden.

$$MEAN: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$MEAN(X) = \sum_{i=1}^{n} X_i * g_i$$

$$wobei \sum_{i=1}^{n} g_i = 1 \; und \;\; g_i \in [0,1] \;\; \forall i \in i..n$$

**Durchschnitt von Positionen (MEAN3D)**. In der Agentensimulation von Navarro (Navarro, 2011) wird bei der Abstraktion die Position einer Agentengruppe als Mittelpunkt der Positionen der beteiligten Agenten gebildet. Bei Hong (Hong, 2013) geschieht dasselbe mit der dreidimensionalen Position von Flugzeugen in einer Formation aus drei Flugzeugen. Auch bei der Simulation von Bodentruppen auf unterschiedlichen Abstraktionsebene muss durch $up$ die (zweidimensionale) Position einer Menge von z.B. Panzern zu einer Einheit durch diese Funktion ermittelt werden (Rabelo, 2015). Wir betrachten hier diese komplexere Variante mit dreidimensionalen Koordinaten.

$$MEAN_{3D}: \mathbb{R}^{3\times n} \rightarrow \mathbb{R}^3$$

$$MEAN_{3D}(X) = \frac{1}{n}\left(\sum_{i=1}^{n} X_{1i}, \sum_{i=1}^{n} X_{2i}, \sum_{i=1}^{n} X_{3i}\right)^T$$

**Maximum (MAX)**. Ohne dies explizit in seinem Beispiel zu verwenden, beschreibt Navarro (Navarro, 2011) auch das Maximum als eine mögliche Abstraktionsfunktion für Eigenschaften von Agenten.

$$MAX: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$MAX(X) = \max_{i \in \{1,\dots,n\}} X_i$$

**Projektion auf eine Komponente (PROJ).** In unseren Untersuchungen (Wittek, 2018) wurde auch ein älteres Beispiel (Wittek, 2016) verwendet. Dabei wird das Seil eines Aufzugs in einer FEM Simulation als Kette von 9 Elementen modelliert. In einem abstrakten, mechanischen Modell des Aufzugs, wird es nur als die Länge des Seiles beschrieben. Die Abstraktionsfunktion ist in diesem Fall die Projektion auf die Position des letzten Elementes der Kette. Analog hierzu wird der Durchsatz einer Gruppe von Maschinen bei Huber (Huber, 2011) als Projektion auf den Durchsatz der letzten Maschine aus der Gruppe modelliert.

$$PROJ: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$PROJ(X) = X_n$$

**Produkt (PROD)**. Auch das Beispiel der Simulation einer Lieferkette aus Kapitel 3 ist relevant. Hier kommt bekanntlich das Produkt als Abstraktionsfunktion zum Einsatz.

$$PROD: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$PROD(X) = \prod_{i=1}^{n} X_i$$

Für die **PROD** Funktion wurden 400 Training- und 4000 Validierungsdaten aus den Input-Daten für das in Kapitel 3 dargestellte Beispiel erzeugt. Mit Hilfe der fünf übrigen Funktionen wurden jeweils 400 weitere Trainings- und 4000 Validierungsdaten erzeugt. Hierzu wurden jeweils gleichverteilte X mit Komponenten aus [0,10] generiert und jeweils mit den Funktionen (**SUM, MEAN, MEAN3D, MAX, PROJ**) gelabelt.

| Name | Dimension der Funktion | Tainingsdaten | Validierungs-daten |
|---|---|---|---|
| **SUM** | $10 \rightarrow 1$ | 400 | 4000 |
| **MEAN** | $3 \rightarrow 1$ | | |
| **MEAN3D** | $3 \times 3 \rightarrow 3$ | | |
| **MAX** | $3 \rightarrow 1$ | | |
| **PROJ** | $10 \rightarrow 1$ | | |
| **PROD** | $3 \rightarrow 1$ | | |

*Tabelle 9.1: Überblick über die Trainings- und Validierungsdatensätze des Benchmarks*

Die so gewonnenen Datensätze bilden die Grundlage für das Benchmark. Tabelle 9.1 gibt einen Überblick über die Dimensionen der Funktionen sowie den Umfang von Trainings- und Validierungsdatensätzen.

### 9.1.2 Regressionsmodelle

Im Folgenden sind kurz die Regressionsmodelle dargestellt, die im Rahmen der Kreuzvalidierung untersucht wurden. Dabei wird eine breite Auswahl an Verfahren betrachtet. Dieses hat jedoch keinen Anspruch auf Vollständigkeit. Auch wurde auf eine detaillierte Parameterstudie der einzelnen Verfahren verzichtet.

**POLY**. Das einfachste verwendete Modell ist ein Polynom n-ten Grades. Im weiteren Verlauf wird mit **POLY2**, **POLY3** usw. ein polynomielles Modell 2-ten bzw. 3-ten Grades bezeichnet. Technisch kann das Modell als lineares Modell behandelt werden. Es kann mit einem einfachen Kleinste-Quadrate-Schätzer (Richter, 2019) gelöst werden.

**GP.** Ein weiteres, sehr verbreitetes Modell für Regression ist die Gaußprozessregression. Dabei wird der Raum von X als unendliche Menge von Zufallsvariablen aufgefasst, welche durch ihren Ort indiziert werden. Y ist dann die Realisation dieser Zufallsvariable. Die Kovarianz zwischen diesen Variablen wird distanzabhängig mit einer Kernelfunktion beschrieben. Im folgenden Benchmark wurde die Summe aus Skalarprodukt und dem *white kernel* als Kernelfunktion verwendet. Das entstehende Modell kann mit der Kleinste-Quadrate-Methode gelöst werden (Rasmussen, 2006). Die Parameter des Kernels werden beim Training ebenfalls optimiert.

**SVR.** Support Vektor Maschinen (SVM) wurden ursprünglich für die Klassifikation entwickelt. Auch hier wird die Kernelfunktion genutzt. Diesmal, um die Daten implizit in einen hochdimensionalen Raum zu transformieren, wo sie nun linear separierbar sind. Auch Support Vektor Maschinen können zur Regression verwendet werden. In diesem Fall spricht man von Support Vektor Regression (SVR) (Vapnik, 1995). Während bei der Gaußprozessregression die Kovarianzen aller Datenpunkte gespeichert werden müssen, werden hier jedoch nur die namensgebenden Supportvektoren gespeichert. Im Rahmen des vorliegenden Benchmarks wurden Modelle mit unterschiedlichen Kernelfunktionen genutzt. Zum Einsatz kommen:

- der lineare Kernel (**SVR-L**),
- der Polynomal-Kernel 3-ten sowie 5-ten Grades (**SVR-P3**, **SVR-P5**),
- der Kernel der radialen Basisfunktion (**SVR-RBF**) und
- ein Kernel auf Basis der Sigmoidfunktion (**SVR-SIG**).

Zur Auswahl des Kernels kommt bei der SVM ein weiterer Verfahrensparameter, der typischerweise mit $\epsilon$ bezeichnet wird. Dieser Parameter kennzeichnet den Bereich um die

Prädiktion, dem beim Lernen ein Fehler von 0 zugewiesen wird. In der vorlegenden Untersuchung wurde dieser Parameter mit 0.03 gewählt.

**RVR.** Das Konzept einer Relevanz Vektor Maschine (RVM) entspricht dem Ansatz der Gaußprozessregression, kombiniert ihn jedoch mit dem Ansatz der Support Vektor Maschine, um nur einige der Datenpunkte zu speichern (Tipping, 2001). Auch hier ist die Nutzung als Regressionsmodell möglich, was zur sogenannten Relevanz Vektor Regression (RVR) führt. Eine Abhandlung über RVM und RVR findet sich dem Buch von Bishop (Bishop, 2006, Kap. 7.2). Auch bei der RVR ist der Kernel, der als Modell der Kovarianz genutzt wird, ein Verfahrensparameter. Hier kommen zum Einsatz:

- der Polynomal-Kernel 3-ten Grades (**RVR-P3**)
- der lineare Kernel (**RVR-L**) und
- ein Kernel der radialen Basisfunktion (**RVR-RBF**).

Die RVR kann ohne den Parameter $\epsilon$ trainiert werden, benötigt dafür jedoch längerer Trainingszeiten, da hier auf einen Ansatz des maximalen Erwartungswertes zurückgegriffen werden muss.

**NN.** Auch neuronale Netze (NN) eigenen sich zur Lösung von Regressionsproblem. Eine Abhandlung über neuronale Netze als Modelle für Regression sowie das hierfür etablierten Trainigsverfahren findet sich beispielsweise dem Buch von Bishop (Bishop, 2006, Kap. 5). Im Benchmark kommen *Rectifier* als Aktivierungsfunktion zum Einsatz. Die Netze werden in 1000 Epochen mit einer Batch-Size von 5 trainiert. Dabei werden Architekturen mit *U* Neuronen pro Layer und L verdeckten Layern eingesetzt. Für die entsprechenden Modelle wurde die Notation NN-UxL (**NN-10x1**, **NN-200x5**) gewählt.

## 9.2 Resultate

Mit den vorgestellten 6 Datensätzen wurde der in Abbildung 9.1 dargestellte Prozess durchlaufen. Dabei wurden die insgesamt 14 Modelle aus dem vorangegangenen Abschnitt genutzt. Diese wurden innerhalb des Schrittes „Kreuzvalidierung“ in der folgenden Reihenfolge durchlaufen:

**POLY2, POLY3, POLY4, POLY5, GP, SVR-RBF, SVR-P3, SVR-P5, SVR-SIG, RVR-L, RVR-RBF, RVR-P3, NN-10x1, NN-200x5**

Diese Reihenfolge ist grob aufsteigend nach der typischen Trainingszeit der Modelle. Um die Auswertung zu beschleunigen, wurde ein Modell, dass bei der Kreuzvalidierung mit seinem $MAE$ den $MAE_{max}$ um den Faktor 10 unterschreitet, direkt ausgewählt. Es wurden die

$MAE_{max}$ Werte 0.2, 0.1, 0.05 und 0.01 untersucht. Tabelle 9.2 fasst die Ergebnisse des Benchmarks zusammen.

| $MAE_{max}$ | 0.2 | | | 0.1 | | | 0.05 | | | 0.01 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **model** | **N** | **MAE** | **model** | **N** | **MAE** | **model** | **N** | **MAE** | **model** | **N** | **MAE** |
| **SUM** | RVR-L | 10 | 0,144 | GP | 20 | 1,33E-07 | GP | 20 | 1,33E-07 | GP | 20 | 1,33E-07 |
| **MEAN** | GP | 10 | 3,47E-07 | GP | 10 | 3,47E-07 | GP | 10 | 3,47E-07 | GP | 10 | 3,47E-07 |
| **MEAN3D** | GP | 10 | 1,33E-06 | GP | 10 | 1,33E-06 | GP | 10 | 1,33E-06 | GP | 10 | 1,33E-06 |
| **MAX** | NN -200x5 | 30 | 0,175 | NN -200x5 | 60 | 0,1 | NN -200x5 | 140 | 0,046 | NN -200x5 | 250 | 0,009 |
| **PROJ** | GP | 10 | 1,23E-05 | GP | 10 | 1,23E-05 | GP | 10 | 1,23E-05 | GP | 10 | 1,23E-05 |
| **PROD** | POLY3 | 20 | 7,97E-08 | POLY3 | 20 | 7,97E-08 | POLY3 | 20 | 7,97E-08 | POLY3 | 20 | 7,97E-08 |

*Tabelle 9.2: Resultate des Benchmarks.*

In der Tabelle werden die vier $MAE_{max}$-Werte gegenübergestellt. Für jeden der Datensätze wird jeweils das Modell, welches ausgewählt wurde, die notwendige Anzahl Beispielen $N$ sowie der mit dem ausgewählten Verfahren erreichte $MAE$ angegeben. Dieser $MAE$ wurde mithilfe des separaten Validierungsdatensatzes bestimmt.

| | **Wertebereich für Labels** | **relativer Fehler zu MAE 0.1** | **MAE Wert zu relativem Fehler 1%** |
|---|---|---|---|
| **SUM** | [0, 100] | 0,1% | 1 |
| **MEAN** | [0, 10] | 1% | 0,1 |
| **MEAN3D** | $(y_1, y_2, y_3)^T$, $y_1, y_2, y_3 \in [0, 10]$ | 1% | 0,1 |
| **MAX** | [0, 10] | 1% | 0,1 |
| **PROJ** | [0, 10] | 1% | 0,1 |
| **PROD** | [0,68; 60,21] | 0,17% | 0,59 |

*Tabelle 9.3: Fehler relativ zum Wertebereich der Datensätze.*

Die Angabe der gewünschten Genauigkeit für das ausgewählte Modell als MAE ist sinnvoll für den Prozess, da dieser Wert eine direkte Bedeutung in der Domäne des Modellierers besitzt. In diesem Benchmark werden jedoch Datensätze unterschiedlicher Wertebereiche verglichen. Deswegen ist hier zudem eine Betrachtung in Bezug auf diesen Wertebereich sinnvoll. Aus diesem Grund werden in Tabelle 9.3 die Wertebereiche der Labels der Datensätze dargestellt. Zudem findet sich exemplarisch für den $MAE$ 0,1 was dieser in Prozent des Wertebereiches bedeutet. Schließlich ist für jeden Datensatz angegeben, welche $MAE$ einem relativen Fehler von 1% entspricht.

Mit diesen relativen Fehlern können nun die absoluten Grenzen aus Tabelle 9.3 in relative Grenzen umgewandelt werden. Eine Aufstellung, der für einen gegebenen, relativen Fehler notwendigen Beispiele, kann Tabelle 9.4 entnommen werden.

| | 2% | | 1% | | 0.5% | | 0.1% | |
|---|---|---|---|---|---|---|---|---|
| **SUM** | RVR-L | 10 | RVR-L | 10 | RVR-L | 10 | GP | 20 |
| **MEAN** | GP | 10 | GP | 10 | GP | 10 | GP | 10 |
| **MEAN3D** | GP | 10 | GP | 10 | GP | 10 | GP | 10 |
| **MAX** | NN-200x5 | 30 | NN-200x5 | 60 | NN-200x5 | 140 | NN-200x5 | 250 |
| **PROJ** | GP | 10 | GP | 10 | GP | 10 | GP | 10 |
| **PROD** | RVR-P3 | 10 | RVR-P3 | 10 | POLY3 | 20 | POLY3 | 20 |

*Tabelle 9.4: Anzahl der Beispiele, um gegebene, relative Fehler zu unterschreiten.*

Es zeigt sich, dass es für alle Datensätze außer **MAX** möglich war, die 0.1% Marke des relativen Fehlers mit kleinen Datensätzen ($N \geq 20$) zu unterschreiten. Zudem zeigt sich, dass das gewählte Gaußprozessmodell gute Ergebnisse für vielen der Datensätze liefert.

Der MAX Datensatz stellt einen massiven Ausreißer dar. Hier werden 250 Beispiel benötigt, um die 0.1% Marke zu erreichen. Zudem ist MAX das einzige Beispiel, in dem ein neuronales Netz die besten Ergebnisse liefert.

## 9.3 Diskussion

Für alle sechs Funktionen aus der Literatur konnte geeignete Modelle gefunden werden. Dabei konnte die 0.1%-Fehlergrenze in fünf Fällen sogar mit äußerst kleinen Trainingsmengen von 20 und weniger Beispielen (für drei Funktionen reichen sogar zehn Beispiele) erreicht werden.

Da neuronale Netze unter Verwendung der Rectifier-Funktion (neben anderen Aktivierungsfunktionen) in der Lage sind, alle Funktionen zu approximieren, ist davon auszugehen, dass zumindest diese Modelle mit hinreichend großen Beispielmengen und unter Verwendung ausreichend großer Trainingszeit in der Lage sind, ein beliebig genaues $up$ zu liefern.

Dennoch zeigt der Benchmark sehr deutlich, dass es Funktionen gibt, die hierzu Beispielmengen benötigen, die so groß sind, dass die automatische Bestimmung für diese Funktionen nicht zweckmäßig erscheint. Wenn für das Erreichen einer großen Genauigkeit sehr viele Beispiele erforderlich sind und der Modellierer mit moderatem Aufwand $up$ explizit beschreiben kann, wird es effizienter sein, $up$ direkt zu beschreiben.

Der Punkt, an dem diese Abwägung kippt, wird im dargestellten Prozess durch $N_{max}$ charakterisiert. In diesem Fall muss der Modellierer die Funktion manuell beschreiben. Dies entspricht einem Rückfall auf den in Kapitel 5 dargestellten Stand der Forschung.

Mit dem Benchmark war es aber auch möglich, zu demonstrieren, dass es zumindest einige Fälle gibt (im Benchmark fünf der sechs Beispiele aus der Literatur) in denen es möglich ist, ein sehr genaues $up$ mit nur wenigen Beispielen zu leren. Mit dieser Möglichkeit geht der vorgestellte Ansatz über den Stand der Forschung hinaus.

# 10 Maschinelles Lernen zur Bestimmung von *down*

Dieses Kapitel konzentriert sich auf die Frage, wie der Algorithmus $down$ automatisch bestimmt werden kann. Wie bereits in Abschnitt 8.2 dargestellt, wird $down$ im Rahmen dieser Arbeit als Algorithmus aufgefasst, dessen Ausgabe einer bedingten Wahrscheinlichkeitsverteilung folgt.

Die Beschreibung eines Regressionsproblems als bedingte Wahrscheinlichkeit ist weit verbreitet und die Basis vieler Verfahren des maschinellen Lernens. Jedoch wird hier vorwiegend nach jenem $y$ gesucht, welches unter einer gegebenen Eingabe die größte Wahrscheinlichkeit aufweist. Für $down$ ist es jedoch notwendig, von der gesamte Verteilung Stichproben zu generieren. Deshalb suchen wir zunächst nach einem Schätzer für bedingte Wahrscheinlichkeitsverteilung. In der englischsprachigen Literatur ist dieses Problem als *conditional density estimation* (CDE) bekannt.

Wir fordern von $down$, die Verteilung der Konkreta aus den Trainingsdaten zu erhalten. Im Allgemeinen ist nicht davon auszugehen, dass es sich hierbei um eine Normalverteilung handelt. Zudem können die Konkreta in mehrere Cluster aufgeteilt sein, wie dies bei der Breite im Lieferkettenbeispiel der Fall ist. Hierdurch verfügt die Verteilung über mehrere Modi. Zudem sind Konkreta mehrdimensional und ihre Komponenten können voneinander abhängig sein. Hierdurch wird aus $P(M^* = k|M = a)$ (vgl. Abschnitt 4.3.2) eine multivariate Verteilung. Auch dies muss durch $down$ abgebildet werden, um die Verteilung der Konkreta zu erhalten.

Somit suchen wir nach einem Schätzer für bedingte, multivariate und mehrmodale Wahrscheinlichkeitsverteilung, der es zudem erlaubt, Stichproben dieser Verteilung zu generieren. In Abschnitt 10.1 werden derartige Verfahren vorgestellt. Dieser Schätzer kann dann genutzt werden, um aus den gelabelten Trainingsdaten diese Verteilung zu schätzen. Innerhalb von $down$ werden dann lediglich Stichproben dieser Verteilung generiert und ausgegeben.

Wie sich zeigen wird, weisen diese Verfahren, aufgrund ihres Aufbaus einen relativ hohen $ARTE$ auf. Tatsächlich ergibt sich bei diesen Verfahren eine untere Schranke für diesen Fehler. Aus diesem Grund wird in Abschnitt 10.2 ein hybrides Verfahren vorgestellt, welches diesen prinzipiellen Mangel umgeht, dabei aber die Qualität der Schätzung der Verteilung erhält.

In Abschnitt 10.3 wird schließlich dieser Ansatz den beschriebenen Verfahren gegenübergestellt.

## 10.1 Conditional Density Estimation

Dieser Abschnitt gibt einen Überblick über bestehende Verfahren zum Schätzen von bedingten Wahrscheinlichkeitsverteilungen. Zentrales Auswahlkriterium für diese Aufstellung ist, dass die Ansätze prinzipiell für die Bestimmung von $down$ eingesetzt werden können.

### 10.1.1 *mixture density networks*

Ein verbreitetes Verfahren zum Schätzen von bedingten Wahrscheinlichkeitsverteilungen sind die *mixture density networks* (MDN) (Bishop, 2013). Die Basis des Verfahrens bildet eine Mischverteilung. Bei einer Mischverteilung wird die Dichtefunktion als gewichtete Summe von k Kernen beschrieben. Bei den MDN kommt hier typischerweise der Gauss-Kern zum Einsatz, weshalb man auch von einem Gaussian Mixture Model (GMM) spricht.

$$P(x) = \sum_{i=0}^{k} \mathcal{N}(\mu_i, \sigma_i) * \pi_i$$

Hierbei ist $\mu_i$ der Mittelwert, $\sigma_i$ seine Varianz und $\pi_i$ das Gewicht des i-ten Kerns. Da wir multivariate Ausgaben $y$ aus $d$ Komponenten betrachten, ist $\mu_i$ jeweils ein Vektor mit $d$ Komponenten. Bei multivariaten Normalverteilungen wird auch die Varianz nicht durch einen Skalar beschrieben, sondern durch eine $d \times d$ Matrix, die sogenannte Kovarianzmatrix, modelliert.

Um mit dem GMM eine bedingte Wahrscheinlichkeit modellieren zu können, werden die Parameter $\mu_i$, $\sigma_i$ und $\pi_i$ als Funktionen in Abhängigkeit von der Eingabe $x$ konstruiert.

$$P(y|x) = \sum_{i=0}^{k} \mathcal{N}\big(\mu_i(x), \sigma_i(x)\big) * \pi_i(x)$$

Der Ansatz der MDN liegt nun darin, diese Funktionen $\mu_i(x)$, $\sigma_i(x)$ und $\pi_i(x)$ mit einem neuronalen Netz zu approximieren. Da die Anzahl der Ausgänge in diesem Beispiel durch die Kovarianzmatrix in Abhängigkeit von $d$ quadratisch zunimmt, wird typischerweise auf eine Vereinfachung zurückgegriffen. Nutzt man als Kovarianzmatrix eine Diagonalmatrix, bleiben lediglich $d$ Freiheitsgrade übrig. Der Nachteil dabei ist jedoch, dass in einem solchen Modell die Kovarianzen der Komponenten der Ausgabe nicht mehr abgebildet werden.

Das so entstehende Netz hat $x$ als Input und $k * d\ (Mittelwerte) + k * d\ (Varianzen) + k(Gewichte)$ Outputneuronen. Um die Gewichte auf die Summe von 1 zu normieren, werden die entsprechenden Outputneuronen mit der Softmax Funktion versehen. Um sicherzustellen, dass die Varianzen positiv sind, wird bei den entsprechenden Neuronen die

Exponentialfunktion genutzt. Die Mittelwerte werden von den entsprechenden Neuronen durchgeleitet.

Für das Training werden nun die Gewichte ω des Netzes als Parameter der drei Funktionen $\mu_i$, $\sigma_i$ und $\pi_i$betrachtet.

$$P(y \mid x, \omega) = \sum_{i=0}^{k} \mathcal{N}\big(\mu_i(x, \omega), \sigma_i(x, \omega)\big) * \pi_i(x, \omega)$$

Für ein Trainingsset $(x_n, y_n) \forall n \in N$ wird nun mittels Backpropagation nach den Parametern gesucht, welcher die negative, logarithmische Likelihood $\mathcal{L}(\omega)$ minimiert.

$$\mathcal{L}(\omega) = -\sum_{n=1}^{N} \ln\left(\sum_{k=i}^{k} \mathcal{N}\big(y_i|\mu_i(x_i, \omega), \sigma_i(x_i, \omega)\big) * \pi_i(x_i, \omega)\right)$$

Um die entstehende Dichtefunktion zu glätten, können die Daten beim Training verrauscht werden (*Noise Regulation*). Zudem kann die Genauigkeit durch Normalisieren verbessert werden. (Rothfuss, 2019).

Es ist möglich, von einem MDN Stichproben zu nehmen. Dies geschieht, in dem man zunächst mit dem Netz zu einem gegebenen Input $x$ die Parameter des GMM ermittelt. Nun wird zufällig und gewichtet mit $\pi$ ein Kern i ausgewählt. Anschließend werden $\mu_i(x, \omega)$ und $\sigma_i(x, \omega)$ genutzt, um die Ausgabe zu generieren.

### 10.1.2 *kernel mixture network*

Ein weiterer, eng verwandter Ansatz sind *kernel mixture networks* (KMN) (Ambrogioni, 2017). Auch hier wird ein GMM als Basis für die Bestimmung der Verteilung genutzt. Allerdings werden die Kerne, wie bei einem Kerndichteschätzer, jeweils fix auf einen der Trainingspunkte gelegt. Um die Anzahl der Gewichte, und somit die Komplexität des Netzes zu reduzieren, kann zudem durch einen Clustering-Ansatz nur eine Teilmenge der Punkte hierzu verwendet werden.

Die Gewichte der Kerne $\pi_i(x, \omega)$ werden weiterhin durch das Netz kontrolliert. Die Varianzen $\sigma_i$ werden entweder, wie bei einem Kerndichteschätzer mithilfe einer vorgegebenen Bandweite definiert oder ebenfalls als Ausgabe des Netzes erzeugt. Letzteres hat sich als überlegen erwiesen (Rothfuss, 2019).

Wie bei MDN kann die negative, logarithmische Likelihood $\mathcal{L}(\omega)$ für das Training des Netzes herrangezogen werden.

Auch KMN profitieren von Noise-Regularisierung und Normalisierung der Trainingsdaten (Rothfuss, 2019).

Wie schon bei MDN können von dem GMM Stichproben erzeugt werden. Mit diesem Mechanismus kann die geschätzte Verteilung prinzipiell als Basis für $down$ genutzt werden.

### 10.1.3 *$\epsilon$-neighbor kernel density estimator*

Ein weiterer Ansatz für das Schätzen einer bedingten Wahrscheinlichkeitsverteilung ist der sogenannte $\epsilon$-*neighbor kernel density estimator* (NKDE) (Rothfuss, 2019; Sugiyama, 2010). Hierbei wird das Prinzip des Kerndichteschätzers auf bedingte Wahrscheinlichkeiten übertragen.

NKDE ist ein sogenannter Lazy Learner (Kidwell, 1997). Hierbei werden alle Trainingspunkte zunächst gespeichert. Für einen Query-Punkt $x^*$ werden dann alle Trainingspunkte $M(x^*, \epsilon)$ ausgewählt, deren $x$-Punkt sich innerhalb einer $\epsilon$-Umgebung um $x^*$ befindet.

$$M(x^*, \epsilon) = \{(x, y): |x - x^* \leq \epsilon|\}$$

Nun wird ein Kerndichteschätzer auf die $y$-Punkte dieser Teilmenge angewandt, indem um jeden der Punkte ein Kern gelegt wird. Die Verteilung ergibt sich wie folgt:

$$P^*(y|x^*) = \sum_{(x,y)=M(x^*,\epsilon)} K(y, h)$$

Dabei ist $K(y, h)$ eine geeignete Kernelfunktion, wie etwa die Dichtefunktion der Standardnormalverteilung.

Die Bandbreite $h$ ist von entscheidender Bedeutung für den Schätzer. Im Falle einer Normalverteilung entspricht er den Varianzen der Verteilung. Diese Bandbreite kann mit der sogenannten Silvermann‘s Rule-of-Thumb (Silverman, 1982) geschätzt werden. Außerdem ist es möglich, mithilfe von Kreuzvalidierung auf den Trainingsdaten eine gute Bandbreite zu suchen. Die Untersuchungen von Rothfuss (Rothfuss, 2019) legen nahe, dass eine Kreuzvalidierung deutlich bessere Ergebnisse liefert.

Um auch dann eine Wahrscheinlichkeit schätzen zu können, wenn sich nur wenige Punkte in der $\epsilon$-Umgebung um $x^*$ befinden, ist es möglich, stattdessen die $k$ nächsten Nachbarn von $x^*$ zu verwenden. Hierdurch wird $k$ – die Anzahl der zu betrachtenden Nachbarn – ein Parameter des Verfahrens.

Um eine Stichprobe für $down$ zu generieren, kann aus den gefundenen Nachbarn zufällig einer ausgewählt werden. Anschließend wird der Kernel auf den Punkt gelegt und eine Stichprobe von diesem Punkt gewählt.

### 10.1.4 Netze mit stochastischen Neuronen

Die bisher vorgestellten Arbeiten haben sich primär damit beschäftigt, die Wahrscheinlichkeitsdichtefunktion zu schätzen. Erst anschließend wurde diese Dichtefunktion genutzt, um eine Stichprobe zu erzeugen. Es gibt jedoch auch Arbeiten, die das Ziel verfolgen, die nicht-determinierte Funktion direkt zu lernen. Ein Beispiel hierfür sind Stochastic Feedforward Neuronal Networks (SFNN) (Tang, 2013)

Als Modell kommt in SFNN ein mehrschichtiges neuronales Netz zum Einsatz. Allerdings werden einige Neuronen durch stochastische, binäre Neuronen ersetzt. Durch die übrigen, deterministischen Neuronen werden realwertige Outputs möglich. Abbildung 10.1 zeigt eine beispielhafte Architektur für ein SFNN. Dabei wird ein eindimensionales X durch drei Hidden-Layer auf ein dreidimensionales Y abgebildet. In den beiden ersten Hidden-Layern kommen dabei jeweils zwei stochastische Neuronen zum Einsatz.

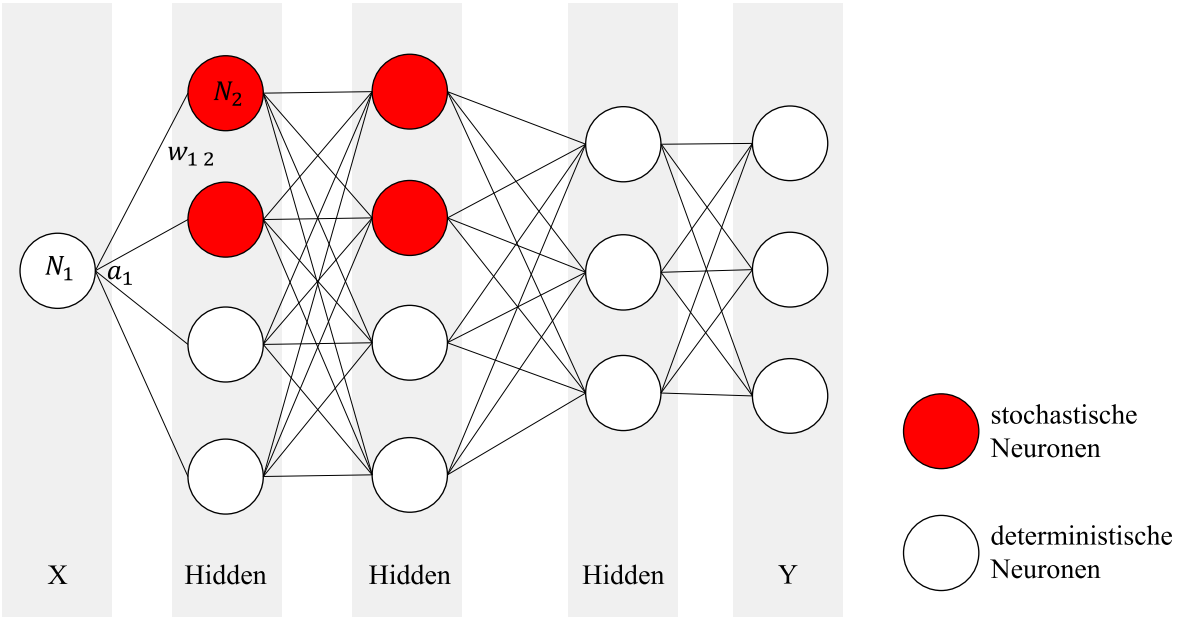



*Abbildung 10.1: Beispielhafte Architektur eines SFNN.*

Diese stochastischen Neuronen können den Wert 1 und 0 annehmen. Die Wahrscheinlichkeit, dass das Neuron $N_i$ den Wert 1 annimmt, wird durch den Wert der Aktivierungsfunktion ermittelt.

$$P(N_i = 1) = f_i\left(\sum_{j<i} a_j w_{ji}\right)$$

Hierbei sind $a_j$ der Ausgabewert des $j$-ten Neurons, $w_{ij}$ das Gewicht der Kante vom $j$-ten zum $i$-ten Neuron und $f_i$ die Aktivierungsfunktion des $i$-ten Neurons. Als Aktivierungsfunktion wird die Sigmoidfunktion genutzt.

$$sig(x) = \frac{1}{1 + e^{-x}}$$

Diese Art der Modellierung von stochastischen Neuronen entspricht der von Sigmoid Belief Networks (SBN) (Neal, 1992), welche ebenfalls in diese Klasse von Netzen fallen. Während SFNN stochastische und deterministische Neuronen mischen, nutzen SBN ausschließlich stochastische Neuronen. Auch SBN nutzen eine Feed Forward Struktur, in der jedes Neuron nur von den vorherigen Neuronen (mit kleinerem Index) abhängen.

Eben diese Feed Forward Struktur ist, was die SBN wiederum von den Boltzman-Maschinen (BM) (Ackley, 1985) unterscheidet. Bei BM sind die stochastischen Neuronen vollständig und bidirektional verbunden. Deren Zustand muss hier als ein Gleichgewicht definiert werden, in das die Neuronen gelangen, wenn sie einer bestimmten Strategie entsprechend feuern.

Während SFNN eine explizite Input-Output-Struktur beschreiben, wird bei BM und SBN nicht zwischen Inputs und Outputs unterschieden. Hier werden dafür sichtbaren Neuronen (Visible, V) und verstecke Neuronen (Hidden, H) definiert. So verfügen die Netze über ein breiteres Funktionsspektrum und können z.B. auch Daten völlig ohne Inputs generieren oder beliebige Teilbelegungen der sichtbaren Neuronen vervollständigen. Um mit BM oder SBN eine nicht-determinierte Funktion abzubilden, werden einige der sichtbaren Neuronen als Inputs definiert. Das Netz vervollständigt dann V, indem es den Output zu einem angelegten Input produziert.

Alle drei Verfahren (SFNN, SBN und BM) teilen sich das Konzept zum Lernen der stochastischen Neuronen. Wie schon bei MDN und KMN wird beim Lernen die log-Likelihood der Trainingsdaten mittels Gradientenverfahrens maximiert. Hierzu werden die partiellen Ableitungen der log-Likelihood nach den Gewichten $w_{ij}$ wie folgt bestimmt (hier für SBN) (Neal, 1992):

$$\frac{\partial L}{\partial w_{ij}} = \sum_{v \in Training} \sum_{s} P(S = s|V = v) * s_i * s_j * sig\left(-s_i \sum_{k<i} s_k w_{ik}\right)$$

Hierbei ist $S$ der Zufallsvektor, der sich aus den Aktivierungen aller versteckten Neuronen ergibt und V der Zufallsvektor, der sich aus den Inputs und den Outputs zusammen ergibt.

Die Wahrscheinlichkeit $P(S = s|V = v)$ ist allerdings nicht bekannt, und muss mithilfe einer Monte-Carlo-Simulation bestimmt werden. Hierzu wird ein X-Trainingspunkt $M$ mal in das Netz gegeben. Das Netz erzeugt nun $M$ Y-Punkte. Die Verteilung dieser Punkte wird geschätzt und für die Bestimmung der Ableitungen $\frac{\partial L}{\partial w_{ij}}$ genutzt. Diese Monte-Carlo-Simulation ist bei jedem Lernschritt zur Bestimmung der neuen Gewichte notwendig.

Durch die Monte-Carlo-Simulation wird das Lernen der drei vorgestellten Netze zu einem Problem. Für BM wird es als „very slow“ (Ackley, 1985) und „painfully slow“ (Neal, 1992, S. 73) beschrieben. Sowohl SBN als auch SFNN verbessern das Lernverfahren teilweise. Allerdings bleibt der grundsätzliche Ansatz, in jedem Lernschritt $M$ Stichproben der aktuell durch das Netz repräsentierten Verteilung zu produzieren, um diese Verteilung zu schätzen, weiterhin bestehen.

### 10.1.5 Zusammenfassung

MDN und KMN mit Noise-Regularisierung und Normalisierung haben sich in der Literatur als gut geeignet herausgestellt, um bedingte Wahrscheinlichkeiten zu modellieren (Ambrogioni, 2017; Bishop, 2013; Rothfuss, 2019). Es ist zu erwarten, dass sie auch bei der Bestimmung von $down$ gute Ergebnisse liefern. NKDE sind in Benchmarks im direkten Vergleich den beiden auf neuronalen Netzen basierten Verfahren unterlegen (Rothfuss, 2019).

SFNN konnten bei multimodaler Regression für einige der Testsets bessere log-Likelihood Werte erreichen (Tang, 2013). Allerdings sind diese Werte dennoch insgesamt mit denen von MDN vergleichbar. Durch die Monte-Carlo-Simulation ist das Lernverfahren von SFNN bei gleicher Epochenzahl im Vergleich zu MDN mindestens mit dem Faktor des Stichprobenumfangs für die Bestimmung der Verteilung (*M)* langsamer. In der Arbeit von Tang wurde beispielsweise $M = 30$ gewählt (Tang, 2013). Aufgrund des viel aufwendigeren Lernverfahrens gegenüber MDN und KMN wurden im Rahmen dieser Arbeit SFNN nicht weiter untersucht.

Das Generieren von Stichproben beruht bei MDN, KMN und NKDE immer darauf, einen der Kerne im GMM auszuwählen und dann von diesem eine Stichprobe zu erzeugen. Allerdings kann dieser Schritt zu einem erhöhten ARTE führen, da diese oft in alle Richtungen streuen. Dieser Zusammenhang wird in 10.4 vor dem Hintergrund des Benchmarks genauer diskutiert werden.

## 10.2 Hybride CDE

In dieser Arbeit wird vorgeschlagen, MDN und NKDE unter Verwendung von $up$ zu verbinden, um die Schätzung der Verteilung und das Erzeugen von Stichproben zu trennen. MDN können so die Verteilung ohne Rücksicht auf den ARTE schätzen und NKDE eine Stichprobe mit beschränktem ARTE generieren.

Abbildung 10.2 gibt eine Übersicht über das hierdurch entstehende hybride Verfahren.

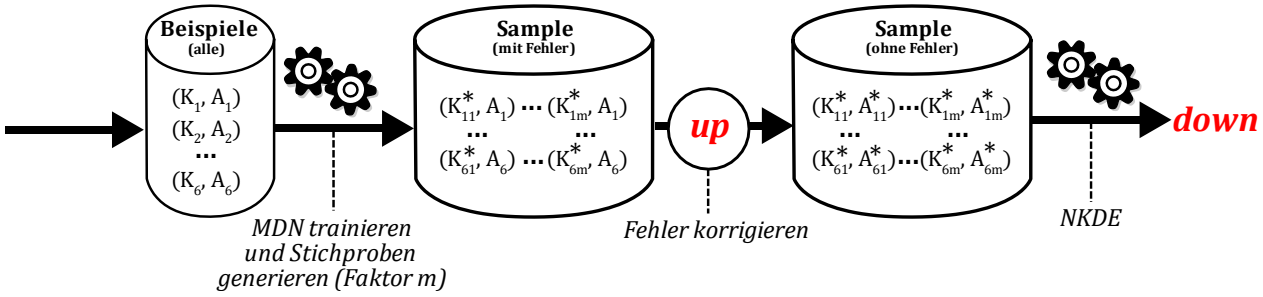



*Abbildung 10.2: Lernvorgang einer hybriden CDE*

Zunächst wird ein MDN genutzt, um die Verteilung der n Beispiele $(K_i, A_i)$ zu lernen (in Abbildung 10.2 n = 6 und somit $i \in [1,6]$). Anschließend werden die Abstrakta der Beispiele $(A_i)$ verwendet, um eine Stichprobe zu erzeugen. Hierzu wird für jedes dieser Abstrakta das MDN m-fach ausgeführt. Es entstehen jeweils m Konkreta $(K_{ij}^*, j \in [1,m])$. Die einzelnen Paare $\left(\left(K_{ij}^*, A_i\right) i \in [1,n], j \in [1,m]\right)$ weisen den für MDN typischen ARTE auf.

Nun wird $up$ auf die Konkreta angewandt, um diesen Fehler zu korrigieren und so den Konkreta in Bezug auf den $ARTE$ korrekte Abstrakta zuzuweisen $(K_{ij}^*, A_{ij}^*)$. Hierbei ist anzumerken, dass es durch den ARTE des MDN so ist, dass diese korrekten Abstrakta sich dennoch vom ursprünglichen $A_i$ unterscheiden. Im Allgemeinen sind also $A_i$ und $A_{ij}^*$ nicht identisch. Entsprechend gibt es auch $m$ unterschiedliche $A_{ij}^*$.

$$A_i \neq A_{ij}^* \wedge A_{ik}^* \neq A_{il}^*$$

$$wobei\ i \in [1,n]\ und\ j,k,l \in [1,m]\ mit\ k \neq l$$

Schließlich wird ein NKDE mit diesen Daten trainiert. Um $down(x)$ zu erzeugen, wird eine Stichprobe mithilfe des NKDE produziert. Dabei wird $h = 0$ verwendet, um keinen zusätzlichen ARTE einzuführen. Der ARTE des Verfahrens ist somit durch den Verfahrensparameter $\epsilon$ beschränkt. Wird bei der Generierung auf die $k$ nächsten Nachbarn von

$x$ zurückgegriffen, kann $\epsilon$ nachträglich als die maximale Entfernung dieser Nachbarn zu $x$ bestimmt werden.

Die Motivation dieser Kombination liegt darin, die Güte des MDN bei der Schätzung der Verteilung mit der hohen Genauigkeit von $up$ zu einem Verfahren mit niedrigem $ARTE$ und niedrigem $MAE$ zu verbinden.

## 10.3 Benchmark

Um die Eignung der in den Abschnitten 10.1 und 10.2 dargestellten Verfahren zur Bestimmung von $down$ zu untersuchen, werden diese im Rahmen des folgenden Abschnittes in Form eines Benchmarks gegenübergestellt.

### 10.3.1 Szenario

Als Szenario kommt im Rahmen dieses Benchmarks das in Kapitel 3 dargestellte Lieferkettenbeispiel zum Einsatz. Als Trainingsdaten werden somit die Inputsets genutzt. Für den Benchmark wurden insgesamt 1000 Trainingspunkte dieser Verteilung erzeugt und mit der beschriebenen Abstraktionsfunktion gelabelt. An dieser Stelle wird zunächst auf ein integriertes Szenario verzichtet, in dem auch $up$ aus Trainingsdaten gelernt wird, um die Eigenschaften der Verfahren zur Bestimmung von $down$ ohne den Fehler, der durch $up$ entsteht, beurteilen zu können. Eine entsprechende, integrierte Evaluation wird in Kapitel 12 durchgeführt.

Die 1000 Trainingspunkte bestehen jeweils aus einem x-Punkt aus $\mathbb{R}$ (Volumen) und einen y-Punkt aus $\mathbb{R}^3$ (Höhe, Breite, Tiefe). Die y-Punkte werden in Abbildung 10.3 dargestellt. Oben sind die Komponenten $y_1$ und $y_2$ dargestellt, unten die Komponenten $y_1$ und $y_3$. Eine Darstellung der Verteilung der x-Punkte kann Abbildung 3.2 entnommen werden. Die y-Punkte bieten eine Reihe von Herausforderungen für das Schätzen ihrer Verteilung.

Die Punkte teilen sich in $y_2$-Richtung in zwei getrennte Cluster auf. Hierdurch muss der Schätzer zwei Modi produzieren.

Die Cluster enthalten unterschiedlich viele Stichproben, entsprechend muss die Verteilung sie unterschiedlich gewichten.

Nur in $y_1$-Richtung sind die Daten normalverteilt, die $y_2$- und $y_3$-Richtung folgen einer Gleichverteilung. Entsprechend sind auch die beiden Cluster jeweils nicht ausschließlich normalverteilt.

Die Komponenten $y_1$ und $y_2$ weisen eine Korrelation auf, die durch die Verteilung abgebildet werden muss.

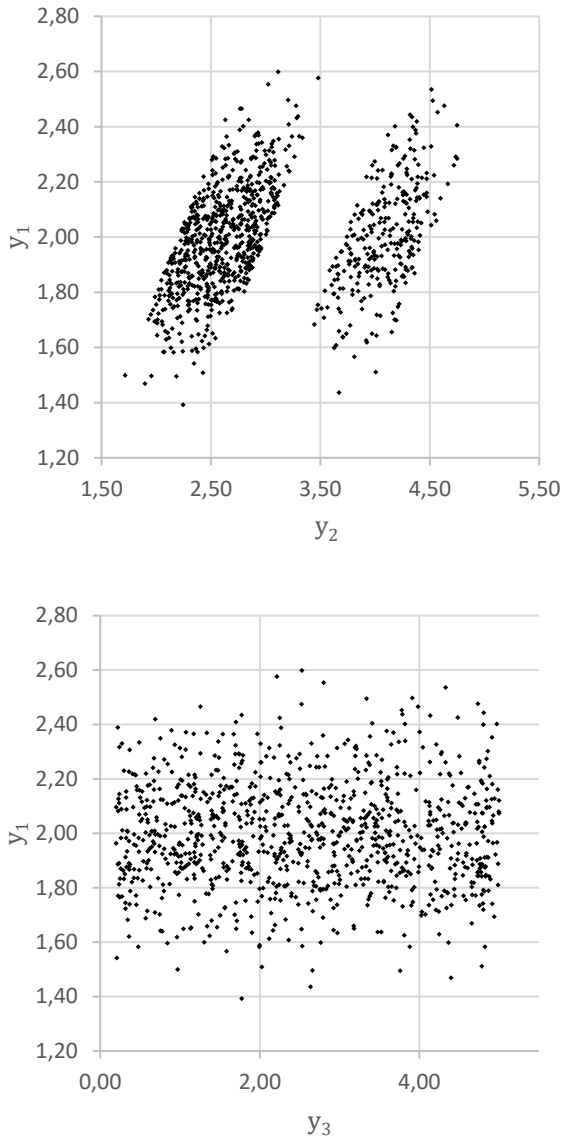



*Abbildung 10.3: y-Punkte der Trainingsdaten des Benchmarks.*

### 10.3.2 Untersuchte Verfahren

In dem Benchmark wurden die folgenden Verfahren mit den jeweils angegebenen Hyperparametern vorgestellt.

**MDN**. Ein *mixure density network* mit den in der Arbeit von Rothfuss (Rothfuss, 2019) ermittelten Hyperparametern. Diese Parameter wurden in einer ausgedehnten Parameterstudie ermittelt.

**MDN-LN**. Ein *mixure density network*. Hier wurde das Rauschen, welches während des Trainings auf die Datenpunkte angewendet wird, reduziert. Dieser Faktor ist eine potenzielle Fehlerquelle.

**MDN-CV3**. Ein *mixure density network*. Die Hyperparameter wurden mit einer dreifachen Kreuzvalidierung ermittelt.

**KMN**. Ein *kernel mixture network* mit den in der Arbeit von Rothfuss (Rothfuss, 2019) ermittelten Hyperparametern.

**KMN-CV3**. Ein *kernel mixture network*. Die Hyperparameter wurden mit einer dreifachen Kreuzvalidierung ermittelt.

**NKDE**. Ein $\epsilon$-*neighbor kernel density estimator*, bei dem die Bandbreite mithilfe der Silvermann‘s Rule-of-Thumb (Silverman, 1982) ermittelt wurde.

**NKDE-CV**. Ein $\epsilon$-*neighbor kernel density estimator*. Die Hyperparameter wurden mit einer dreifachen Kreuzvalidierung ermittelt.

**HYBRIDE**. Der in Abschnitt 10.2 beschriebene, hybride Schätzer. Das MDN wurde wie in **MDN-CV3** konfiguriert.

Tabelle 10.1 liefert einen Überblick über die verwendeten Hyperparameter.

| Verfahren | *Epsilon* | *Bandbreite* | *Architektur des Netzes* | *Anzahl der Kerne* | *Trainings-Epochen* | *x Rauschen* | *y Rauschen* |
|---|---|---|---|---|---|---|---|
| **MDN** | | | (16,16) | 20 | 1000 | 0.2 | 0.1 |
| **MDN-LN** | | | (16,16) | 30 | 1500 | 0.01 | 0.01 |
| **MDN-CV** | | | (16,16) | 10 | 1000 | 0.1 | 0.1 |
| **KMN** | | | (16,16) | 50 | 1000 | 0.2 | 0.1 |
| **KMN-CV** | | | (16,16) | 200 | 1000 | 0.15 | 0.2 |
| **NKDE** | 0,4 | [0.09360802 0.35728374 0.6750336 ] | | | | | |
| **NKDE-CV** | 0,3 | [0.18058997, 0.07433021, 0.09862794] | | | | | |
| **HYBRIDE** | 0 | 0 | (16,16) | 10 | 1000 | 0.1 | 0.1 |

*Tabelle 10.1: Übersicht über die Hyperparameter der im Benchmark betrachteten Verfahren.*

Die Implementierung des Benchmarks basiert auf einer Bibliothek für CDE, die ebenfalls von Rothfuss vorgestellt wurde (Rothfuss, 2019).

### 10.3.3 Bewertungskriterien

Aus den Anforderungen (siehe 7.10) für $down$ werden die im Folgenden beschriebenen Bewertungskriterien abgeleitet.

Als ein Maß für die Einhaltung der Konsistenzanforderung wird der Absolut Round Trip Error ($ARTE$) verwendet werden. Zur Berechnung von ARTE wurde $up_0(\mathrm{y}_1, \mathrm{y}_2, \mathrm{y}_3) = \mathrm{y}_1 * \mathrm{y}_2 * \mathrm{y}_3$ aus Abschnitt 3.10.4 verwendet.

Um den Grad, mit der $down$ die Verteilung P nachbildet, und somit die Verteilungskonformität erfüllt, zu bestimmen, kommen zwei unterschiedliche Kriterien zum Einsatz.

Die durchschnittliche log-Likelihood ist ein Maß dafür, wie wahrscheinlich die Testdaten unter Annahme eines trainierten Modells sind.

$$LL(VAL, P^*) = \frac{1}{|VAL|} \sum_{(x,y) \in VAL} log\big(P^*(y|x)\big)$$

Hierbei ist $VAL$ eine unabhängige Menge von Datenpunkten, die nicht zum Lernen des Modells genutzt wurden. Die durchschnittliche log-Likelihood ist geeignet, um die Qualität der gelernten Verteilung verschiedener Modelle für die gleichen Trainings- und Testdaten zu vergleichen (Bishop, 2006; Rothfuss, 2019; Tansey, 2016). Für die Bewertung wird die durchschnittliche log-Likelihood auf einem Validierungsset mit 10000 Punkten verwendet.

Allerdings bietet die durchschnittliche log-Likelihood kein absolutes Gütemaß. Deshalb wird für den Benchmark zudem ein statistischer Test verwendet. Der sogenannte multivariate, nicht parametrische Cramer-Test (Baringhaus, 2004) erlaubt es, für zwei Stichproben zu prüfen, ob sie derselben Grundgesamtheit entstammen. Der Test ist in der Lage, hierfür eine Wahrscheinlichkeit anzugeben, den p-Wert. Der p-Wert, der im Rahmen des Benchmarks angegeben wird, basiert jeweils auf einen Stichprobenumfang von 1000 Punkten.

### 10.3.4 Resultate

In diesem Abschnitt werden die Ergebnisse des Benchmarks vorgestellt. Tabelle 10.2 gibt einen Überblick über die Resultate der verschiedenen Methoden.

In Hinblick auf die Verteilung zeigt sich, dass sowohl MDN, MDN-LN, MDN-CV als auch HYBRIDE einen p-Wert $\geq$ 0,99 erreichen. Dies bedeutet, dass der Test mit mehr als 99-prozentiger Sicherheit davon ausgeht, dass die Konkreta, die mit den Verfahren generiert wurden, denen der Trainingsdaten entsprechen. Diese Verfahren erreichen somit eine Fehlerwahrscheinlichkeit von deutlich weniger als 5%.

| *Verfahren* | *Cramer-Test p-Wert* | *∅ log-Likelihood* | *∅ ARTE* |
|---|---|---|---|
| ***MDN*** | ***0,9999*** | *-0,2099* | *1,8223* |
| ***MDN-LN*** | ***0,9980*** | *-0,4405* | *0,7311* |
| ***MDN-CV*** | ***0,9949*** | *0,0047* | *1,4932* |
| ***KMN*** | *0,6923* | *-0,5683* | *3,2555* |
| ***KMN-CV*** | *0,8462* | *-0,1196* | *2,6971* |
| ***NKDE*** | *0,1239* | *-2,0611* | *4,0009* |
| ***NKDE-CV*** | *0,9141* | *-1,0273* | *1,3122* |
| ***HYBRIDE*** | ***0,9900*** | *-0,4406* | ***0,0109*** |

*Tabelle 10.2: Übersicht über die Ergebnisse des Benchmarks.*

KMN-CV und NKDE-CV erreichen einen p-Wert ≥ 0,80. KMN und NKDE erreichen lediglich einen p-Wert ≤ 0,70. Somit erreichen diese Verfahren eine Fehlerwahrscheinlichkeit von deutlich über 5%. Die log-Likelihood entspricht im Wesentlichen dieser Gruppierung. MDN, MDN-LN, MDN-CV und HYBRIDE erreichen hier Werte ≥ -0,5. Zudem fällt KMN-CV in diese Gruppe. Der Cramer-Test ist abhängig vom Stichprobenumfang. Sind die Stichproben zu klein, kann der Test keine Aussage treffen. Entsprechend ist hier der p-Wert meist kleiner. Aus diesem Grund findet sich in Abbildung 10.4 ein Überblick über den p-Wert in Abhängigkeit vom Stichprobenumfang.

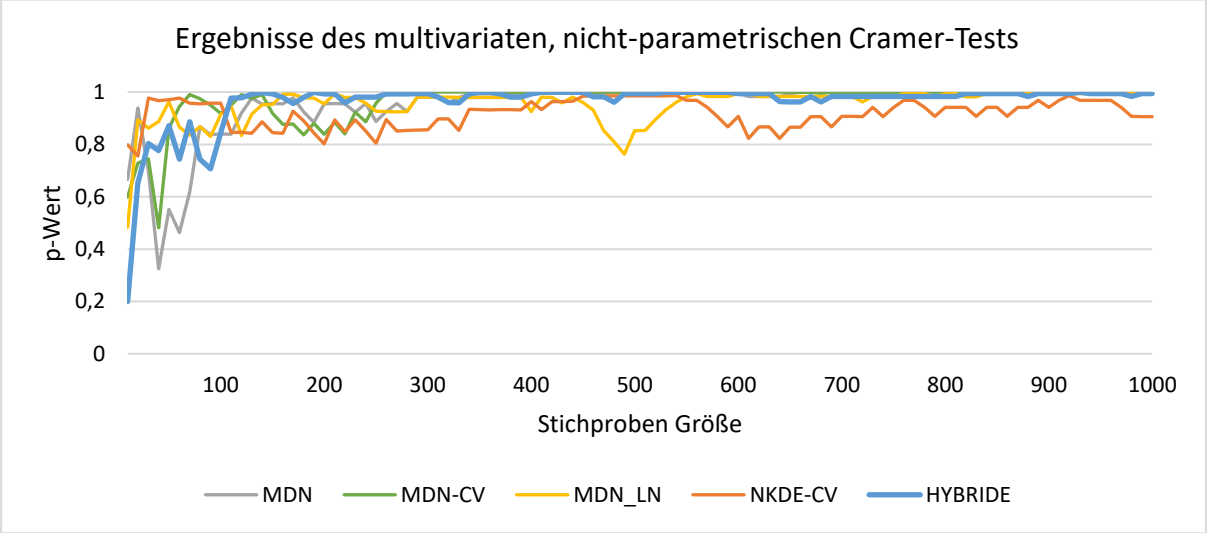



*Abbildung 10.4: p-Wert der Top 5 Verfahren in Abhängigkeit vom Stichprobenumfang*

In der Grafik sind nur die Verfahren mit einem p-Wert von mehr als 0.9 dargestellt (MDN, MDN-CV, MDN-LN, NKDE-CV und HYBRIDE). Ab einem Stichprobenumfang von 150 liegen alle auf MDN mit Noise-Regularisierung basierenden Modelle bis zum vollen Stichprobenumgang oberhalb der 0.95 Marke. Lediglich bei MDN_LN, bei denen die Noise-Regularisierung nur in geringerem Umfang angewandt wurde, gibt es einen kleinen Ausbruch bei ungefähr 500.

HYBRIDE weist mit 0.01 den mit Abstand geringsten ARTE auf. Fast alle Verfahren haben einen ARTE von mehr als 1. Lediglich MDN_LN erreicht mit 0,73 einen ARTE, der etwas unter 1 liegt.

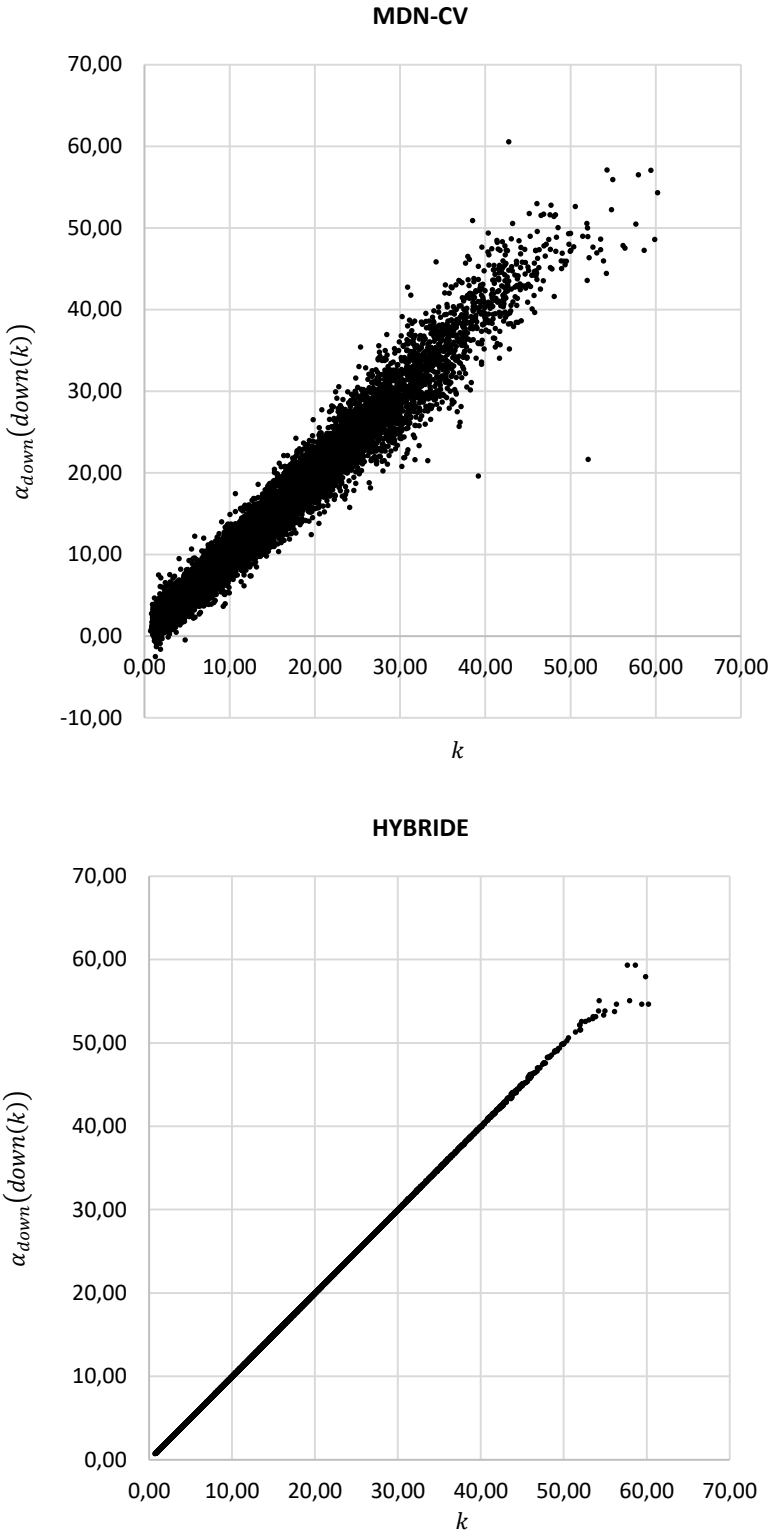



*Abbildung 10.5: Darstellung des ARTE von MDN-CV und HYBRIDE*

Abbildung 10.5 verdeutlich den Unterschied zwischen dem $ARTE$ von MDN-CV (1,4932) und HYBRIDE (0,01). Auf der x-Achse ist dabei $k$ abgetragen (also die Abstrakta der Trainingsdaten), während auf der y-Achse $\alpha_{down}\big(down(k)\big)$ dargestellt wird. Während die Punkte bei MDN-CV gut erkennbar von der Diagonalen abweichen (auf der Diagonalen ist ARTE = 0), treffen die Punkte bei HYBRIDE diese Diagonale fast exakt. Lediglich ganz am oberen Rand der Skala, bei Werten, die größer als 50 sind, ist eine Streuung erkennbar. Im oberen Diagramm für MDN-CV kann man zudem erkennen, dass einige Pakete mit einem kleinen, negativen Volumen entstanden sind. Dies geschieht, obwohl es im kleinen Bereich zwischen 1 und 5 mehr als 160 Trainingspunkte gibt (vgl. Abbildung 3.2).

## 10.4 Diskussion

Insbesondere die *mixture density networks* (MDN) haben sich innerhalb des Benchmarks als gut geeignet für das Schätzen der Verteilung erwiesen.

In Hinblick auf $ARTE$ ergibt sich für MDN, KMN und NKDE eine systematische Schwäche. Diese lässt sich durch den Aufbau dieser Verfahren mithilfe eines GMM erklären. Letztlich beruht das Generieren von Stichproben immer darauf, einen der Kerne im GMM auszuwählen und dann von diesem eine Stichprobe zu erzeugen. Allerdings wird dieser Schritt zu einem erhöhten ARTE führen.

Nehmen wir an, das GMM wählt für einen Query-Punkt $k$ einen Kern mit Mittelpunkt $\mu$ aus, der genau in der gewünschten Äquivalenzklasse $[k]_{\sim}$ mit $\alpha(k) = a$ liegt. Der ARTE dieses Mittelpunktes wäre entsprechend 0.

Durch die Varianz der ausgewählten Normalverteilung wird dieser Punkt verrauscht und liegt möglicherweise nicht mehr in der Äquivalenzklasse. Es entsteht ein Fehler.Für unser Beispiel kann dieser Fehler leicht abgeschätzt werden. Zur Veranschaulichung verzichten wir dabei auf die Breite als dritte Dimension und wählen $\alpha\big((h,t)\big) = h * t$.

Die Normalverteilung nutzt eine diagonale Kovarianzmatrix. Die Komponenten werden dadurch als unabhängig betrachtet. Somit kann die Varianz des Produktes der beiden Variablen wie folgt bestimmt werden (Goodman, 1960).

$$Var(h * t) = E(h)^2 Var(t) + E(t)^2 Var(h) + Var(h)Var(t)$$

Definieren wir nun $\sigma = \min(\{Var(h), Var(t)\})$.

$$Var(h * t) \geq E(x_1)^2\sigma + E(x_2)^2\sigma + \sigma^2$$

$$= \sigma * (E(x_1)^2 + E(x_2)^2 + \sigma) \geq \sigma * (E(x_1)^2 + E(x_2)^2)$$

Die Erwartungswerte sind im Beispiel durch $E(h) = 2$ und $E(t) = 2.6$ gegeben. Durch Einsetzten erhält man:

$$\sigma * 10.76$$

Man kann also abschätzen, dass $\alpha_{down}((h,t))$ mit einer Varianz von mehr als $\sigma * 10.76$ streuen würde.

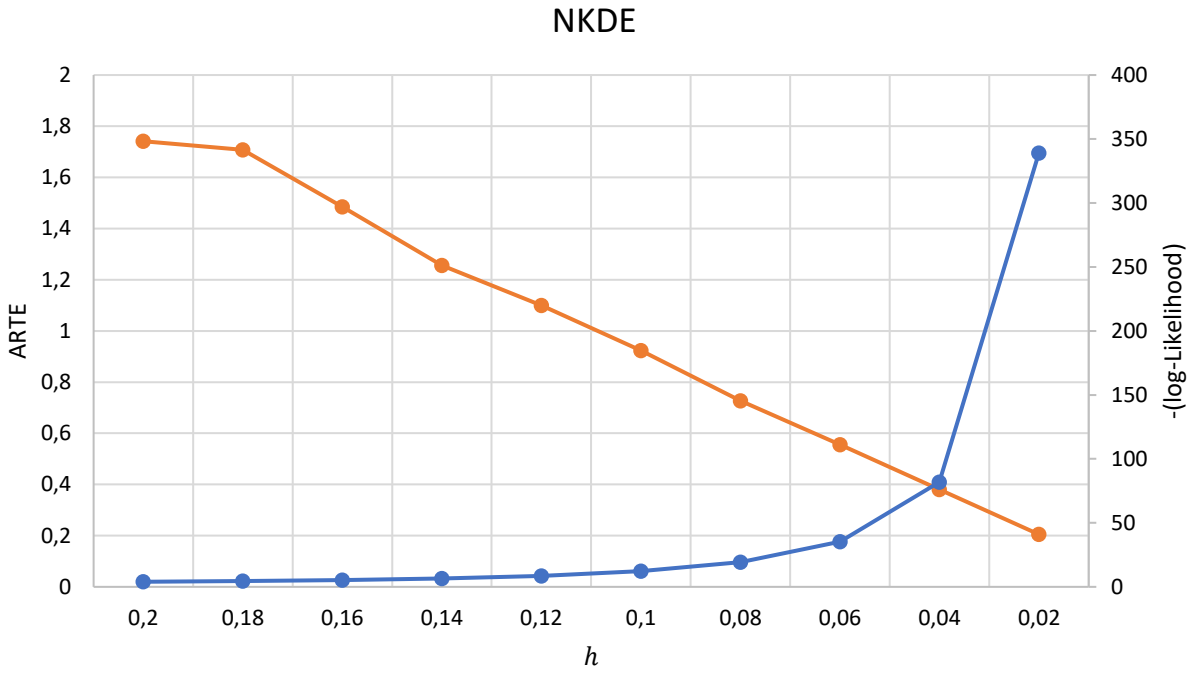



*Abbildung 10.6: Auswirkungen der Wahl von $\sigma$ auf NKDE.*

Auch der Versuch, ein möglichst kleines $\sigma$ zu erzwingen, löst das Problem nicht, da dies massiv die Verteilung stört, die von $down$ erzeugt wird.

Abbildung 10.6 verdeutlich diesen Zusammenhang anhand des Lieferkettenbeispiels. Wir nutzen hierzu NKDE da hier die Varianz direkt über die Bandweite $h$ eingestellt werden kann. Da ARTE ebenfalls vom gewählten $\epsilon$ abhängt, wurde $\epsilon = 0$ gewählt. So nutzt der Schätzer immer die k nächsten Nachbarn des Querypunktes. (Im Beispiel wurde k = 10 gewählt)

Auf der x-Achse ist $h$ abgetragen. Auf der y-Achse werden $ARTE$ (links) und negative log-Likelihood (rechts) abgetragen. Sowohl ARTE als auch die negative log-Likelihood sollten möglichst klein sein. Während $ARTE$ mit sinkendem $h$ ebenfalls kleiner wird, erhöht sich die negative log-Likelihood.

Dieser Benchmark legt nah, dass HYBRIDE die guten Eigenschaften der MDN in Hinblick auf die Verteilung der Konkreta, die von $down$ erzeugt werden, beibehält und den $ARTE$ drastisch reduzieren kann.

Im vorgestellten Benchmark ist HYBRIDE in der Lage, Stichproben zu erzeugen, für welche der Cramer-Test mit mehr als 99-prozentiger Sicherheit bestätigt, dass sie dieselbe Verteilung aufweisen wie die Trainingsdaten. Gleichzeitig liegt der ARTE bei lediglich 0,01.

Zudem kann mithilfe des $\epsilon$-Paramters des NKDE der ARTE beschränkt werden. Hierdurch können Fehler, wie das Auftreten von negativen Paketvolumina im Benchmark, verhindert werden.

# 11 Multi-Level-Simulationsplattform

Im Rahmen dieser Arbeit wurde eine Multi-Level-Simulationsplattform prototypisch umgesetzt. Sie folgt der in Kapitel 4 vorgestellten, formalen Architektur und erlaubt so, diese später anhand von Fallstudien zu untersuchen. Die Umsetzung basiert auf Java (Java, 2020) und nutzt die, ebenfalls auf Java basierende, Simulationsumgebung Ptolemy (siehe 2.2.2) zur Integration verschiedenster Simulationssemantiken. Für das Lernen von $up$ und $down$ kommen Python-basierte Implementierungen zum Einsatz. Die gelernten Algorithmen $up$ und $down$ werden gespeichert und von der Plattform für die Durchführung einer Multi-Level-Simulation geladen und ausgeführt.

Im folgenden Kapitel werden die Plattform und ihre Schnittstellen in Relation zur Simulationsumgebung und zu den gelernten Algorithmen dargestellt. Abbildung 11.1 gibt einen Überblick über den Aufbau der Plattform in Form eines UML-Klassendiagrammes.

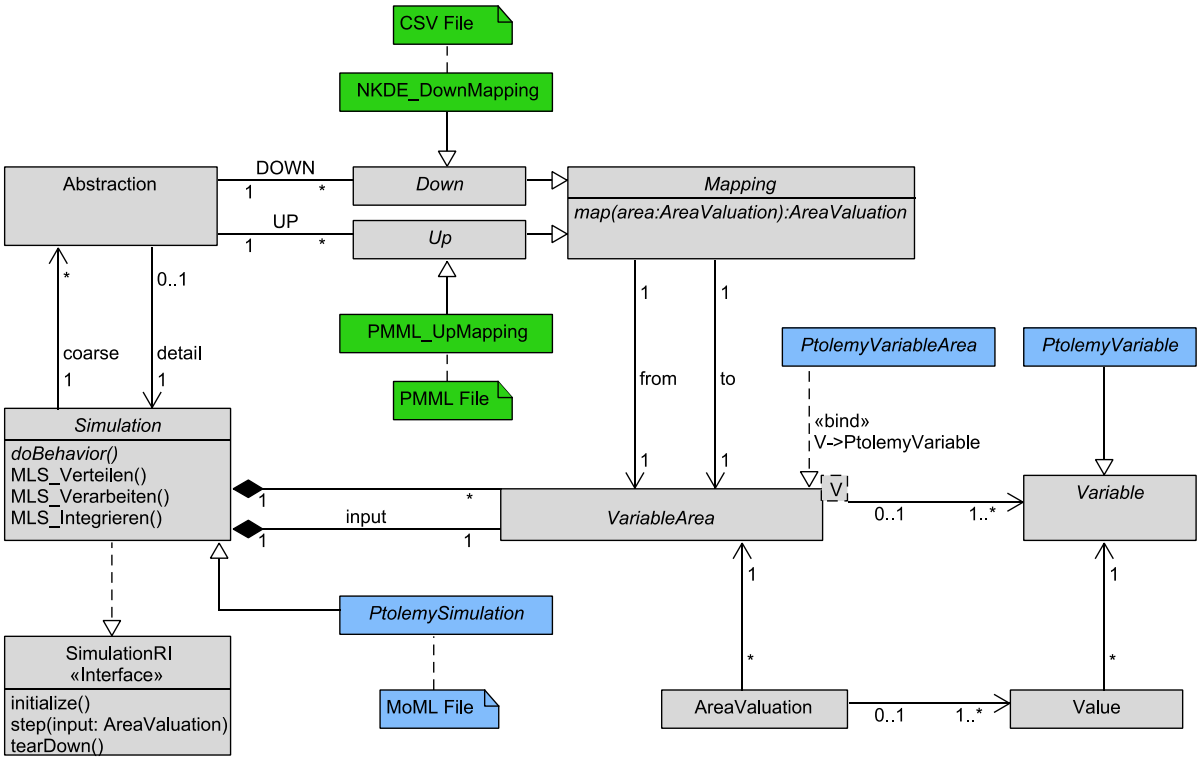



*Abbildung 11.1: UML-Klassendiagramm der Multi-Level-Simulationsplattform*

Dabei werden die Kernelemente der Plattform in Grau dargestellt. Diese können auf die Strukturen des formalen Modells abgebildet werden. Näheres hierzu findet sich in Abschnitt 11.1. Die in Blau dargestellten Klassen sind für die Integration der Ptolemy-Simulationsumgebung notwendig. Diese Klassen werden in Abschnitt 11.2 näher erläutert.

Zudem finden sich in Grün jene Klassen, die zur technischen Integration der gelernten Algorithmen $up$ und $down$ notwendig sind. Sie werden in Abschnitt 11.3 behandelt.

## 11.1 Umsetzung des formalen Modells

Die Plattform ist eine Umsetzung des in Kapitel 4 dargestellten, formalen Modells einer Multi-Level-Simulation. In weiten Teilen wurden die Elemente dieses Modells direkt in Klassen der Plattform umgewandelt. Teilweise gibt es kleinere, technisch motivierte Abweichungen. In diesem Abschnitt wird daher die Verbindung zwischen dem formalen Modell und der Implementierung der Plattform hergestellt.

**Variable.** Die Klasse Variable entspricht der Menge $VARIABLE$ aus dem formalen Modell. Jede Instanz dieser Klasse entspricht dem Namen einer Variablen. Wie im Modell können diesen Variablen Werte zugewiesen werden.

$$x: V \rightarrow DOMAIN$$

Dabei ist $DOMAIN$ auf den Wertereicht des Java Datentyp *double* beschränkt. Somit wird im Rahmen dieser Implementierung *double* als atomarer Datentyp eingesetzt.

**VariableArea.** Für die Umsetzung der Variablenabstraktion aus dem Modell werden Variablenbereiche (VariableArea) eingeführt. $ABSTRACTION_{C,A}$, verfügt mit $C$ und $A$ indirekt über einen Mechanismus, um Mengen von Variablen zu selektieren. Für die Plattform muss dieser Mechanismus explizit umgesetzt werden. Alle Variablen einer Simulation werden daher eindeutig einer `VariableArea` zugewiesen. Die `VariableArea` sind dabei disjunkt.

**AreaValuation.** Die Belegung der Variablen einer Simulation mit Werten darf nur auf Ebene eines gesamten Variablenbereiches geschehen. Entsprechend nutzen alle Schnittstellen der Simulation vollständige Variablenbereichsbelegungen (`AreaValuation`) als Parameter oder Rückgabewerte. An dieser Stelle ist anzumerken, dass die Simulation selbst bei der Durchführung eines Simulationsschrittes uneingeschränkt auf seine Variablen zugreifen kann.

**Mapping**. Die abstrakte Klasse `Mapping` wird als Oberklasse für $up$ und $down$ eingesetzt. Die Klasse hat Assoziationen zu zwei Variablenbereichen. Der Bereich `from` ist der Ursprung der Belegungen, welche dann mit der `map`-Methode in Belegungen des Variablenbereiches `to` umgewandelt werden. Somit verbindet jedes Mapping zwei Variablenbereiche miteinander.

**Up und Down**. Die Klassen `Up` und `Down` erben von `Mapping`. Die Nutzung der beiden Klassen stellt eine marginale Abweichung vom formalen Modell dar.

Im formalen Modell wird die Umwandlung der Belegung eines vollständigen Lesebereiches $\overline{R}$ in die Belegungen eines vollständigen Schreibbereich $\underline{W}$ mit einem monolytischen $down$ beschrieben. In der Implementierung kann diese Umwandlung auf mehrer `Down` Klassen aufgeteilt werden.

Im Lieferkettenbeispiel wird die Umwandlung der Bestellungen durch eine andere Instanz der `Down`-Klasse realisiert als die Abbildung der Pakete. Dies wird im formalen Modell durch ein einzelnes $down$ zusammengefasst, welches beide Umwandungen nebeneinander durchführt.

Gleiches gilt für die Umwandlung von Belegungen des Lesebereiches $\underline{R}$ in Belegungen des Schreibbereiches $\overline{W}$ mit einem monolytischen $up$ im formalen Modell. Diese kann in der Implementierung in mehrere `Up`-Klassen unterteilt werden. Diese Abweichung ändert nicht das Verhalten der Simulation, gestaltet aber die Implementierung übersichtlicher.

**Abstraction**. Die Klasse `Abstraction` verbindet zwei Simulationen in den Rollen einer Grob- (`coarse`) und einer Detail- (`detail`) Simulation miteinander. `Abstraction` werden über die Assoziationen `UP` und `DOWN Mappings` der Subklassen `Up` und `Down` zugewiesen. Wie bereits erwähnt, können hierbei die Lese- und Schreibbereiche ($\underline{R}$ und $\overline{W}$ sowie $\overline{R}$ und $\underline{W}$) durch mehrere separate `Up` und `Down` unterteilt werden.

**SimulationRI.** Die vorliegende Implementierung ermöglicht die verteilte Ausführung der Multi-Level-Simulation mithilfe des Java-RMI Framework. Das Interface `SimulationRI` erbt von dem `Remote` Interface aus diesem Framework. Hierdurch kann die `Simulation`-Klasse (siehe unten), die dieses Interface implementiert, in einer RMI-Registry veröffentlicht werden. Somit können die Simulationen auf unterschiedlichen Hardwareknoten parallelisiert werden. Die verteilte Ausführung ist für die Simulationen transparent.

In Abbildung 11.1 werden dabei nur die drei zentralen Methoden dargestellt. Hinzu kommen noch eine Reihe von Komfortmethoden, sowie Methoden, um das Instanzgeflecht aus `Abstraction` und `VariableAreas` sowie deren Wertebelegung zu verwalten. Diese sind jedoch für die hier gegebene Übersicht nicht relevant.

Die Methode `initialize` versetzt die Simulation in ihren Startzustand. `step` führt einen Multi-Level-Schritt aus (Siehe Abschnitt 7.7). Da im Rahmen dieses Multi-Level-Schrittes auch ein Simulationsschritt der betreffenden Simulation ausgeführt wird, müssen hier Belegungen der Inputvariablen übergeben werden. `tearDown` beendet die Simulation und gibt etwaige beanspruchte Ressourcen wieder frei.

**Simulation**. Die Klasse `Simulation` schließlich realisiert das $MODEL$ aus der formalen Architektur (Abschnitt 7.1).

$$\boldsymbol{MODEL} \coloneqq \underbrace{\mathbb{P}(INPUT)}_{\boldsymbol{Inputvariablen}} \times \underbrace{\mathbb{P}(STATE)}_{\boldsymbol{Zustandsvariablen}} \times \underbrace{\mathbb{P}(\widehat{STATE})}_{\boldsymbol{Startzustand}}$$

Die Input- sowie die Zustandsvariablen werden hier als Assoziationen zu Variablenbereichen, welche wiederum Mengen von Variablen zusammenfassen, realisiert. Der Startzustand wird durch Implementierung der `initialize` Methode gesetzt.

Die Klasse `Simulation` implementiert die `step`-Methode des Interfaces `SimulationRI`. Hierzu werden die in Abschnitt 7.8 dargestellten Teile der Berechnungsvorschrift, *Verteilen*, *Paralleles Verarbeiten* und *Integration* durchgeführt. Dabei ruft `step` nacheinander die folgenden drei privaten Methoden auf:

`MLS_Verteilen` – Die Methode iteriert alle assoziierten Abstraktionen und alle hiermit assoziierten `Down`-Abbildungen. Sie führt diese aus und integriert die so generierten Detailbelegungen in die entsprechenden Detailsimulationen.

`MLS_Verarbeiten` – Innerhalb dieser Methode wird `step` auf allen angeschlossenen Detailsimulationen aufgerufen. Zudem wird die eigene `doBehavior`-Methode der Simulationen ausgeführt. Alle diese Aufrufe geschehen in unterschiedlichen, parallelen Threads. `MLS_Verarbeiten` wartet bis alle Threads abgeschlossen sind. Durch die Verwendung des RMI-Frameworks können diese Aufrufe auch durch unterschiedliche Hardwareknoten realisiert werden. Hierdurch wird eine effiziente, verteilte Simulation möglich.

`MLS_Integrieren` – Abschließend werden die Abstraktionen und deren `Up` Abbildungen integriert und ausgeführt. Die entstehenden, neuen Belegungen werden wieder in die Simulation integriert.

## 11.2 Ptolemy Integration

Die Multi-Level-Simulationsplattform integriert Ptolemy II. Da es eine der zentralen Zielsetzungen des Ptolemy-Projektes ist, die Heterogenität von Modellen mit unterschiedlichen Ausführungssemantiken zu überwinden, finden sich in Ptolemy eine Vielzahl von fundierten und erprobten Werkzeugen, um die verschiedensten Modellsemantiken zu integrieren (Eker, 2003). Durch die Integration in die Plattform werden die Werkzeuge für die vorliegende Implementierung nutzbar. Die Integration in die Plattform wird durch die folgenden Klassen realisiert:

Die Klasse `PtolemySimulation` erbt von `Simulation`. Die Modelle von Ptolemy werden in Form einer MoML genannten XML Syntax gespeichert (Lee, 2000). Die Klasse nutzt

die Bibliotheken von Ptolemy, um ein Modell dieses Formates zu initiieren. Die API von Ptolemy erlaubt es nun, das Modell schrittweise auszuführen, beliebige Zustandsteile zu ändern und sogar die Struktur des Modells anzupassen.

Da für die Plattform definiert sein muss, was die Semantik eines Ausführungsschrittes ist, muss das Modell über einen definierten Top-Level-Direktor verfügen. Der Diskret Time (DT)-Direktor entspricht hierbei genau dem Zeitmodell aus der Architektur (siehe Abschnitt 7.2). Hierbei vergeht bei jedem Aufruf eine feste Zeitspanne $\Delta t$. Diese Länge entspricht der Intervalllänge aus der Architektur. Aus diesem Grund muss der Top-Level-Direktor in der von `PtolemySimulation` geladenen MoML Datei immer der DT-Direktor sein. Jeder Aufruf der Methode `doBehavior` führt zu einem voranschreiten der Simulation um diese Schrittweite.

Diese Einschränkung entspricht dem theoretischen Betrachtungsrahmen dieser Arbeit, ist aber praktisch wenig limitierend. Wie bereits erwähnt, ist es ein zentrales Ziel von Ptolemy, unterschiedliche Ausführungsmodelle aufeinander abzubilden. Entsprechend gibt es eine Vielzahl von Arbeiten, welche die formalen Grundlagen dafür legen, unter dem DT-Direktor verschiedenste andere Ausführungssemantiken zu integrieren.

Die Klassen `PtolemyVariable` und `PtolemyVariableArea` realisieren den Zugriff auf den Zustandsraum der Ptolemy-Simulationen. Auf der Seite des Ptolemy-Modells werden die für eine Multi-Level-Simulation relevanten Variablen explizit mithilfe eines entsprechend MLS-Variable-Aktors markiert. Technisch wäre es möglich, über die API jedes Element des Zustandsraumes der Simulation zu lesen und zu schreiben. Die Lösung mit expliziten Variablen-Aktoren vereinfacht jedoch die spätere Konfiguration einer Multi-Level-Simulation. Sie erlaubt es, Variablen separat zu benennen und diese in der Plattform mit diesem Namen zu adressieren. Die Variablen können dabei auch Felder des Ptolemy-Records sein. Diese erlauben es, im Modell einzelne, primitive Datentypen zu bündeln.

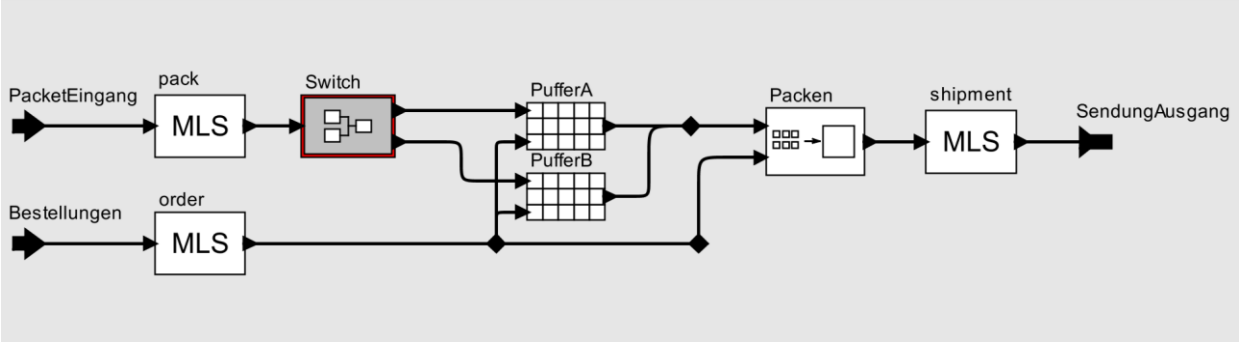



*Abbildung 11.2: Beispiel für die Nutzung von MLS-Variablen.*

Abbildung 11.2 zeigt am Bespiel der detaillierten Kommissionierung, wie die MLS-Variablen genutzt werden können. In der Abbildung werden die eingehenden Pakete (*pack*) sowie Bestellungen (*order*) und die ausgehenden Sendungen (*shipment*) für eine Multi-Level-Simulation nutzbar gemacht.

Ptolemy verfügt über einen gesonderten Wert für abwesende Signale (⊥). Auch im Beispiel ergibt es Sinn, dass nicht in jedem Zeitschritt ein Paket in der Kommissionierung eingehen muss, weshalb dieser Wert auch in den Modellen des Beispiels genutzt wird.

Um ⊥ in Java abbilden zu können, ist jedoch ein Umweg notwendig. Um die Struktur von Record-Typen beim Lesen in die Plattform nicht zu verlieren, muss für jede MLS-Variable ein strukturierter *null*-Wert definiert werden. Die Variable wird mit diesem Nullwert belegt, wenn sie ⊥ erhält.

Ist beispielsweise kein Paket am Ausgang, liest die Plattform nicht `shipment = ⊥`, sondern `shipment = {A=null, H=null, B=null, T=null}`. Wird umgekehrt die Variable aus der Plattform heraus auf diesen strukturierten *null*-Wert gesetzt, gibt die MLS-Variable den ⊥ Wert aus.

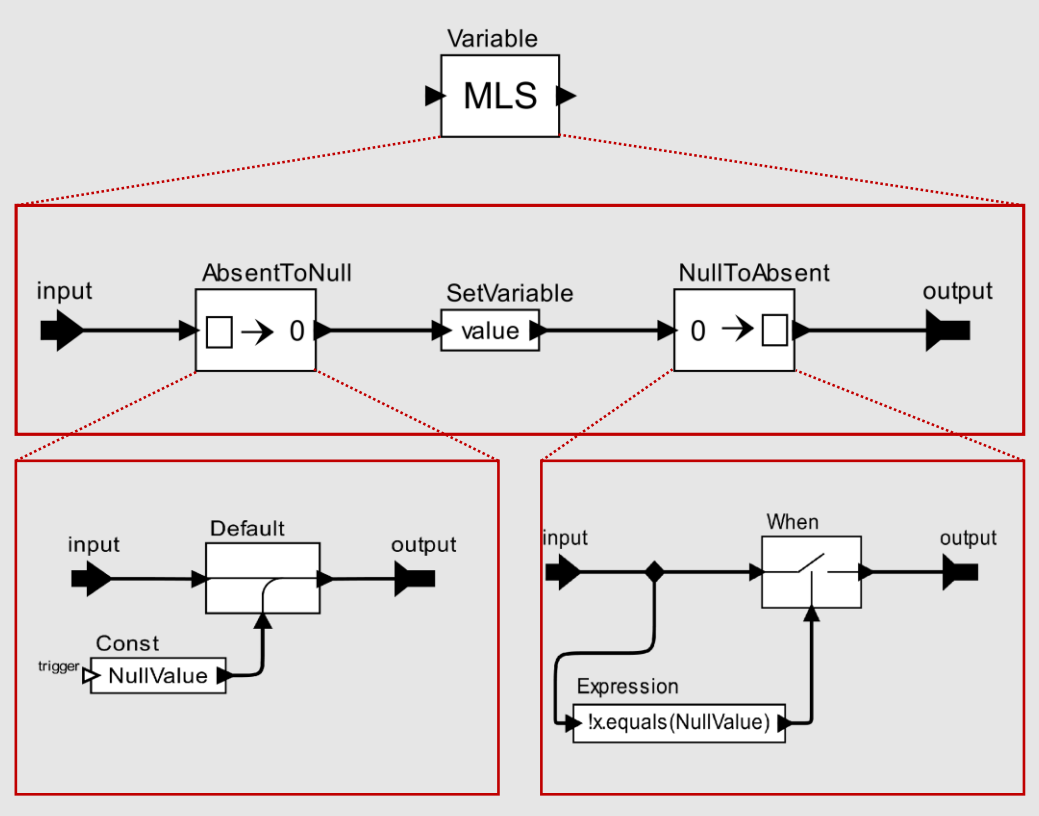



*Abbildung 11.3: Funktion der MLS-Variable.*

Abbildung 11.3 stellt dar, wie der MLS-Variablen-Aktor aufgebaut ist. Er verfügt über die beiden Aktoren AbsentToNull und NullToAbsent. Diese verwenden die atomaren Aktoren *Default* bzw. *When* des Ptolemy-Frameworks, um diese Funktionalität zu realisieren.

## 11.3 Integration der gelernten Algorithmen *up* und *down*

Die Integration von *up* erfolgt über die Klassen `PMML_UpMapping`. Wie bereits im Benchmark aus Kapitel 9 wird auch in der Plattform für das Training eine auf Python basierende Implementierung eingesetzt. Diese wendet den in Abbildung 9.1 dargestellten Prozess zur Bestimmung von *up* an. Anschließend wird das gefundene, nun im Arbeitsspeicher befindliche, Modell mithilfe der Bibliothek SkLearn2PMML (SkLearn2PMML, o. J.) in das Predictive Model Markup Language (PMML, o. J.) Austauschformat überführt. Auf diesem Weg die Modelltypen *POLY2, POLY3, POLY4, POLY5, NN-10x1* sowie *NN-200x5* exportiert werden.

Die Klasse `PMML_UpMapping` lädt dieses Modell mithilfe der Bibliothek JPMML-Evaluator („JPMML-Evaluator - Java Evaluator API for PMML“, o. J.). Während einer Multi-Level-Simulation werden Aufrufe der `map`-Methode durch das geladene Modell ausgeführt.

Die Integration von *down* erfolgt über die Klasse `NKDE_DownMapping`. Auch hier erfolgt das Training innerhalb eines Python-Skriptes. Dabei wird, wie bei der hybriden CDE vorgeschlagen (siehe Abbildung 10.2), zunächst ein MDN trainiert. Anschließend kann eine Stichprobe (*Sample mit Fehler*) erzeugt werden. Diese Stichprobe wird als CSV-Datei exportiert und mithilfe des trainierten *up* korrigiert (*Sample ohne Fehler*). Die korrigierte Stichprobe wird erneut als CSV Datei gespeichert. Innerhalb der Klasse `NKDE_DownMapping` wird schließlich eine Java-seitige NKDE-Implementierung trainiert. Aufrufe der `map`-Methode werden an diesen NKDE weitergeleitet.

Da es sich bei dem NKDE um einen Lazy-Learner handelt, müssen die Trainingspunkte, also die *Sample ohne Fehler*, gespeichert werden. Bei einer Anfrage werden dann alle Konkreta innerhalb eines bestimmten Umkreises um ein Query-Konkretum gesucht. Die einfachste Implementierung hierfür stellt eine lineare Suche dar. Allerdings empfiehlt es sich, hier mit Indexierungsverfahren zu arbeiten. In der Plattform wurde die ELKI-Bibliothek genutzt (ELKI, o. J.). Hierbei werden verschiedene Indizierungsverfahren für derartige Nachbarschaftsanfragen unterstützt. Die Plattform verwendet die kd-Trees (Bentley, 1975) aus der Implementierung von ELKI.

# 12 Evaluation

Zusammengenommen stellen die in Kapitel 4 vorgestellte, formale Architektur einer Multi-Level-Simulation sowie das in Kapitel 8 darauf aufbauend definierte Konzept einer lernenden Multi-Level-Simulation eine Antwort auf die erste Forschungsfrage dar.

1. **Wie kann eine Architektur für lernende Multi-Level-Simulation aussehen?** Diese Architektur muss insbesondere die Konsistenzanforderung aus 4.3.1 sicherstellen. Zudem muss sie einen Rahmen für das Lernen von $up$ und $down$ liefern und dabei insbesondere festgelegt, wie $up$ und $down$ beschrieben werden, sodass sie im Rahmen der Architektur eingesetzt werden können.

Die jeweils in dem Kapitel 9 und 10 durchgeführten Benchmarks der Lernverfahren für $up$ und $down$ liefern eine Antwort auf die Forschungsfragen 2 und 3.

2. **Welche Methode des maschinellen Lernens eignet sich zur Bestimmung von $up$?** Hierzu muss die Methode anhand weniger Beispielen die Abstraktionsbeziehung $\alpha$ nachbilden.

3. **Welche Methode des maschinellen Lernens eignet sich zur Bestimmung von $down$?**
   Hierzu muss die Methode aus den modellierten Inputs die bedingte Wahrscheinlichkeit aus dem Verteilungskriterium 4.3.2 nachbilden.

Dieses Kapitel beschreibt eine integrierte Evaluation des gefundenen Konzeptes einer lernendenden Multi-Level-Simulation. Diese ist erforderlich, um schließlich auch die 4. Forschungsfrage beantworten zu können.

4. **Fügen sich die Lernverfahren für Algorithmen $up$ und $down$ tatsächlich mit der Architektur zu einer Multi-Level-Simulation zusammen?** Neben der isolierten Evaluation der beiden Lernverfahren ist hierzu eine Evaluation der lernenden Multi-Level-Simulation im Rahmen von Evaluationsszenarien notwendig.

Hierzu werden zwei Fallstudien durchgeführt.

In einer ersten Fallstudie wird das in Kapitel 3 dargestellt Lieferkettenbeispiel als lernende Multi-Level-Simulation umgesetzt (Abschnitt 12.1).

In einer zweiten Fallstudie wird ein Beispiel aus der Evaluation des Ansatzes von Baohong (Baohong, 2007) eingesetzt. Das Beispiel befasst sich mit einer Formation von Flugzeugen über feindlichem Gebiet. Es findet sich zudem in leicht abgewandelter Form in der Arbeit von (Hong, 2013) wieder. Hier wird es zur Evaluation von *multi-resolution modeling spaces*

genutzts. Abschnitt 12.2 beschreibt die Anwendung der lernenden Multi-Level-Simulation auf dieses Beispiel.

## 12.1 Fallstudie 1 – Lieferkette

In dieser Fallstudie wird das vorgeschlagene Lösungskonzept auf das in Kapitel 3 dargestellte Beispiel der Entwicklung einer Lieferkette angewandt.

Hierzu werden die in Kapitel 3 dargestellten Modelle mithilfe der Multi-Level-Simulationsplattform (siehe Kapitel 11) integriert. Um den Ablauf der Multi-Level-Simulation untersuchen zu können, werden die Modelle um entsprechende Logging-Elemente erweitert. Hiermit soll insbesondere die Konsistenz der entstehenden Multi-Level-Simulation im Hinblick auf die Konsistenzanforderung (siehe Abschnitt 4.3.1) untersucht werden.

Zur Bestimmung von $up$ und $down$ kommen die in den Kapiteln 9 und 10 beschriebenen Verfahren zum Einsatz. Diese bereits isoliert untersuchten Lernverfahren werden im Rahmen dieses Szenarios in der vom Lösungskonzept vorgeschlagenen Abhängigkeit verwendet. Wie in Kapitel 8 beschrieben wird dabei zunächst $up$ bestimmt und anschließend für die Erzeugung von Labels bei der Bestimmung von $down$ genutzt.

Das auf diese Weise verkettet ermittelte $down$ wird zudem in Hinblick auf die Verteilung der Konkrete, wie sie in der Verteilungskonformitätsanforderung (siehe Abschnitt 4.3.2) thematisiert wird, untersucht. Hierfür kommt erneut der Cramer-Test zum Einsatz.

### 12.1.1 Struktur

Betrachten wir zunächst die Struktur der Multi-Level-Simulation, wie sie im Rahmen dieser Fallstudie umgesetzt wurde. In den Abschnitten 3.3 und 3.5 wurde das Grobmodell einer Lieferkette und das Detailmodell der Kommissionieranlage innerhalb dieser Lieferkette vorgestellt.

Für die Integration in einer Multi-Level-Simulation wurden diese Modelle um MLS-Variablen erweitert. Zudem wurde nach jeder Variablen ein Logging-Aktor integriert. Dieser schreibt für jeden Wert, der die MLS-Variable verlässt, einen Eintrag in eine Log-Datei. Abbildung 12.1 zeigt die beiden Modelle nach diesen Änderungen.

In das Modell der Lieferkette (Abbildung 12.1 oben) wurden vier MLS-Variablen integriert. Die Variable `pack` speichert abstrakte Pakete, welche vom Lager in die Kommissionierung übertragen werden. Diese setzen sich aus einer Identifikationsnummer (ID), einem Adressaten (A) sowie dem Volumen (V) zusammen. Die Variable `order` speichert Bestellungen, welche auf diesem Weg an die Kommissionierung übergeben werden. Diese setzt sich aus einem

Adressaten (A) sowie einer Liste der bestellten Pakete (ITEMS) zusammen. Beide Variablen zusammen stellen den Lesebereich $\bar{R}$ des Grobmodells dar.

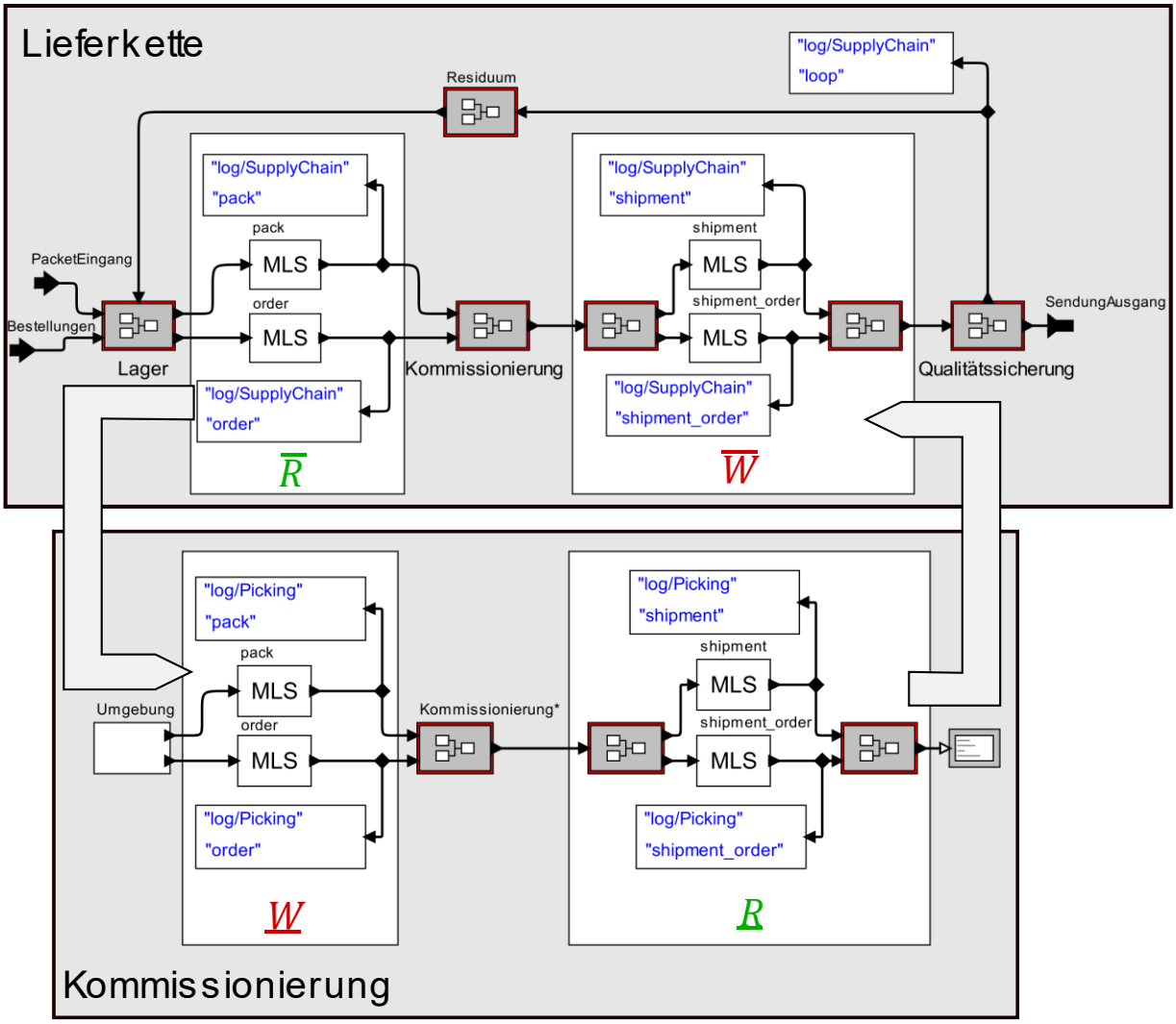



*Abbildung 12.1: Darstellung von Lese- und Schreibbereich beider Modelle.*

Die Variablen `shipment` und `shipment_order` beschreiben Sendungen, welche die Kommissionierung verlassen und zur Qualitätssicherung übertragen werden. Dabei besteht `shipment` aus einer Identifikationsnummer (ID), einem Adressaten (A) sowie einem Volumen (V). Die Variable `shipment_order` enthält eine Kopie der zu dieser Sendung gehörenden Bestellung (A, ITEMS). Die beiden zusätzlichen Aktoren vor und hinter den Variablen der Sendung sind notwendig, da die Kommissionierung beide Variablen in einem einzelnen Ausgang zusammenfasst. Die Variablen bilden zusammengenommen den Schreibbereich $\bar{W}$ der Grobsimulation.

Auch in das Modell der Kommissionierung wurden vier MLS-Variablen eingefügt. Auch dies ist in Abbildung 12.1 dargestellt (unten). Dabei beschreiben `pack` und `shipment` die

konkreten Varianten der gleichnamigen Variablen im Grobmodell. Beide Variablen setzen sich wiederum aus einer Identifikationsnummer (ID), einem Adressaten (A) sowie den Dimensionen (H,B,T) zusammen. Die Variablen `order` und `shipment_order` sind identisch zu ihren Gegenstücken aus dem Modell der Lieferkette.

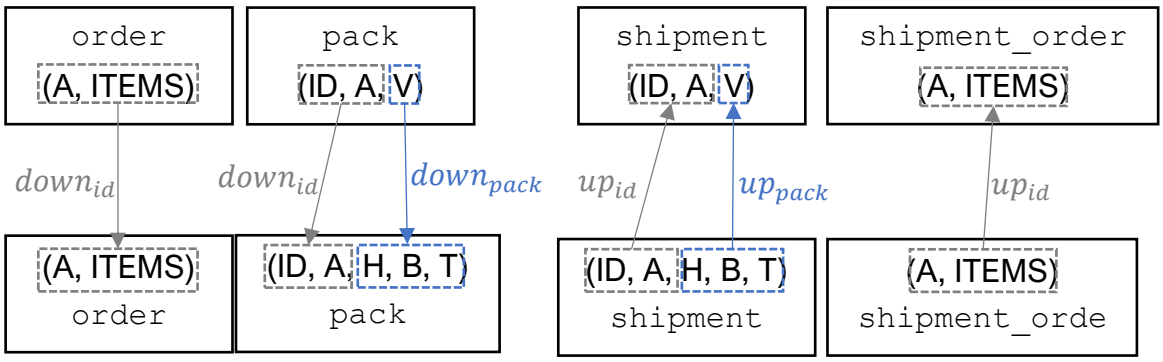



*Abbildung 12.2: Zusammenhang zwischen den Elementen der Variablen.*

Abbildung 12.2 fasst noch einmal zusammen, aus welchen Elementen die insgesamt acht Variablen bestehen. Dabei werden die Variablen der Grobsimulation erneut oben dargestellt, während die Variablen der Detailsimulation unten dargestellt werden.

Zudem stellt die Grafik dar, wie die Variablen durch die Multi-Level-Simulation miteinander verbunden sind. Dabei stehen $up_{id}$ und $down_{id}$ für die Identitätsabbildung. Die MLS-Plattform verfügt über vordefinierte $up$- und $down$-Klassen, welche diese triviale Abbildung umsetzen. Das Volumen des abstrakten Paketes (V) soll durch $down_{pack}$ auf die Dimensionen (H, B, T) des konkreten Paketes abgebildet werden. Die Dimensionen der Sendung (H, B, T) soll von $up_{pack}$ wiederum auf das Volumen (V) der abstrakten Sendung abgebildet werden. Die $up_{pack}$ und $down_{pack}$ werden im nächsten Abschnitt ermittelt.

### 12.1.2 Bestimmen von *up* und *down*

Die Bestimmung von $up_{pack}$ folgt dem in Kapitel 9 vorgeschlagenen Vorgehen. Abbildung 12.3 gibt einen Überblick über den konkreten Ablauf.

Der $MAE_{max}$ für dieses Szenario wird mit 0,01 gewählt. Wie bereits in Tabelle 9.3 dargestellt, nimmt das Volumen im Lieferkettenbeispiel einen Wert zwischen 0,68 und 60,21 ein. Somit entsprich ein $MAE_{max}$ von 0,01 einem relativen Fehler von 0,17 ‰. Nun werden zunächst 10 zufällig ausgewählte, detaillierte Pakete der 1000 Input-Pakete (siehe Abbildung 3.5) manuell mit Volumina gelabelt. In der Kreuzvalidierung liefert das Modell **POLY3** die geringsten,

durchschnittlichen Fehler und erreicht bei dem Training mit allen 10 Punkten einen MAE von 0,715. Trainiert mit allen Traingspunkten erreicht das Modell jedoch nur 0,428

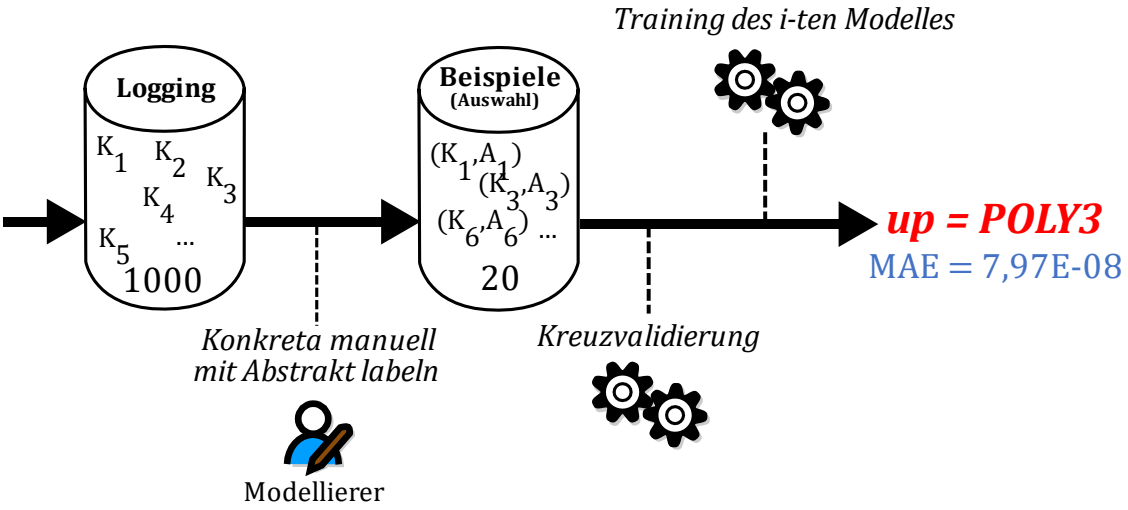



*Abbildung 12.3: Ablauf der Bestimmung von up.*

Da dieser Wert über $MAE_{max}$ liegt, werden 10 weitere, zufällige, detaillierte Pakete mit einem Volumen versehen. Bei der Kreuzvalidierung liefert erneut **POLY3** als Modell den geringsten MAE (0,438). Der MAE beim Training mit allen 20 Punkten liegt bei 7,97E-08 und somit weit unter dem geforderten Wert. Entsprechend wird das trainierte Modell als $up_{pack}$ verwendet. In Tabelle 12.1 ist dieser Ablauf zusammengefasst.

| **Anzahl der Beispiele** | POLY2 CV | POLY3 CV | POLY4 CV | POLY5 CV | NN-10x1 CV | NN-200x5 CV | **Final Training** |
|---|---|---|---|---|---|---|---|
| **10** | 0,903 | 0,438 | 0,715 | 1,215 | 8,486 | 2,665 | **POLY3 0,428** |
| **20** | 0,185 | 0,097 | 0,502 | 0,758 | 4,826 | 3,869 | **POLY3 7,97E-08** |

*Tabelle 12.1: Zusammenfassung des Trainings von up.*

Nun kann $up_{pack}$ für das automatische Labeling eingesetzt werden. Dabei wird der in Kapitel 10 dargestellte Prozess durchlaufen. Der resultierende Ablauf wird in Abbildung 12.4 zusammengefasst.

Hierbei werden die insgesamt 1000 detaillierten Pakete (*Logging*) zunächst durch $up_{pack}$ mit Volumina gelabelt. Die resultierende Trainingsmenge (*Beispiele*) weist gegenüber dem Orakel einen $MAE$ von 7,57E-08 auf.

Diese Menge wird nun genutzt, um ein MDN zu trainieren. Mit diesem MDN werden dann insgesamt 20000 Sample erzeugt. Hierzu wird das trainierte MDN 20-mal auf die 1000 Konkreta aus der Beispielmenge angewandt. Diese Menge weist nun einen $MAE$ von 0,75 (gegenüber dem Orakel) auf.

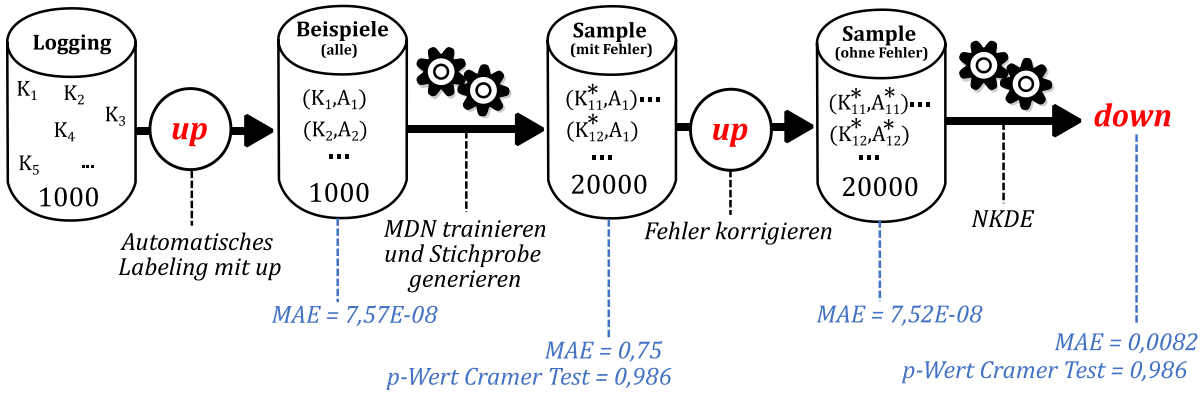



*Abbildung 12.4: Ablauf bei der Bestimmung von down.*

Dieser Fehler wird mithilfe von $up_{pack}$ korrigiert und schließlich wird der NKDE trainiert. Dieser NKDE wird als $down_{pack}$ genutzt. Er erreicht im Cramer-Test, welcher die Verteilung der durch diesen Schätzer generierten Daten mit der Verteilung der ursprünglichen Trainingsdaten (Logging) vergleicht, einen p-Wert von 0,986. In Bezug auf die Validierungsmenge aus Abschnitt 10.3.3 mit 10000 Datenpunkten erreicht das $down_{pack}$ einen durchschnittlichen ARTE von 0,0082.

### 12.1.3 Durchführung

Die dargestellte Multi-Level-Simulation wurde über 25000 Zeitschritte ausgeführt. Dabei enthielten die Inputs der Simulation der Lieferkette insgesamt 564 Eingangspakete sowie Bestellungen für 113 Sendungen, in welchen alle dieser Pakete wieder abgerufen wurden. Nach Durchlauf der Simulation hatten 113 Sendungen die Lieferkette verlassen.

Für alle Belegungen der 8 MLS-Variablen wurden Log-Einträge erstellt. Um den Zusammenhang zwischen diesen Logs und der Einhaltung der Konsistenzanforderung zu beleuchten, wollen wir zunächst einen der Schritte genauer betrachten. Es handelt sich um den Schritt von t=165 zu t=166. Er wird in Abbildung 12.5 dargestellt. In der Log-Datei der

Lieferkette finden wir zum Zeitstempel 165 einen Eintrag mit dem Paket ID=14, welches das Volumen von 9,4206 aufweist[2].

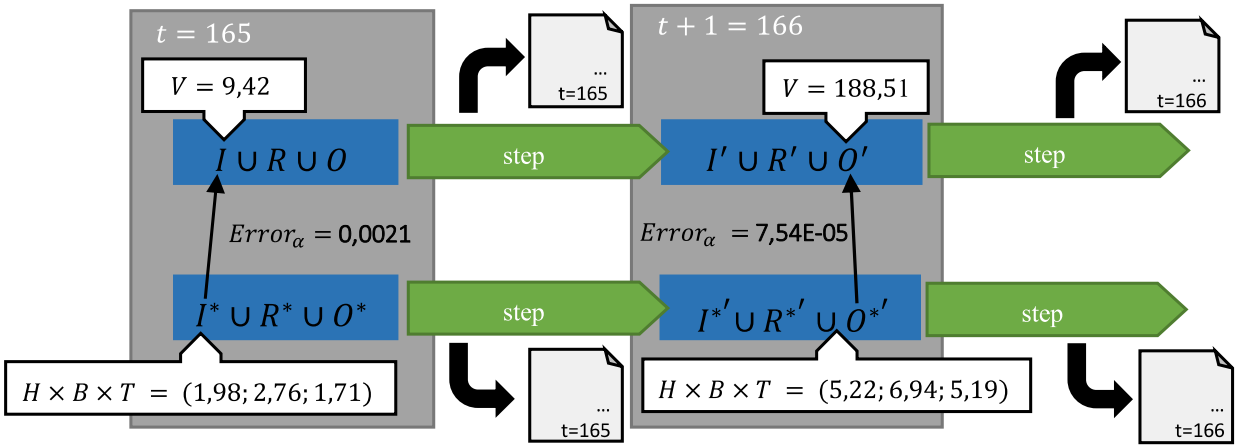



*Abbildung 12.5: Beispiel eines Simulationsschrittes (t=165).*

Da der Log-Eintrag während des Schrittes von 165 nach 166 geschrieben wurde, wissen wir, dass die MLS-Variable in diesem Simulationsschritt das entsprechende Paket ausgegeben hat. Somit war die Variable V aus $I$ direkt vor dem Simulationsschritt mit diesem Wert belegt. Auch für die Detailsimulation gibt es einen entsprechenden Eintrag mit dem Zeitstempel 165. Hier findet sich das detaillierte Paket ID = 14 mit den Dimensionen (1,9857; 2,7616; 1,7175) (gerundet). Somit war $I^*$ vor dem Simulationsschritt mit diesem detaillierten Paket belegt. Somit wird $\alpha$ (die wahre Variablenabstraktion) mit einem Fehler von 0,0021 eingehalten ($Error_\alpha$).

Aus den darauffolgenden Log-Einträgen mit dem Zeitstempel t=166 wissen wir, dass in diesem Zeitschritt zudem eine Sendung die Kommissionierung verlassen hat. In der Log-Datei der Kommissionierung findet sich die detaillierte Sendung ID = 1001 und die Dimensionen (5,2260; 6,9416; 5,1966) mit dem Zeitstempel 166. In der Log-Datei der Lieferkette findet sich die abstrakte Sendung 1001 mit dem Volumen 188,5166 mit diesem Stempel. Der $Error_\alpha$ beträgt hier gerade einmal 7,55E-05.

Somit kann für den gesamten Zeitschritt 166 festgestellt werden, dass die Konsistenzanforderung, bis auf den jeweiligen Fehler, eingehalten wird. Entsprechend können wir den MAE sowie den ARTE, der sich zwischen den Abstrakta und Konkreta aus $I$ und $I^*$

[2] Alle Werte wurden auf 4 Nachkommastellen gerundet. In der Grafik wurde zur Erhöhung der Übersicht auf 2 Nachkommastellen gerundet. Sämtliche Berechnungen wiesen eine Genauigkeit von 11 Nachkommastellen auf.

bzw. $O$ und $O^*$ zeigt und in den Log-Dateien aufgezeichnet wird, als Maß für die Einhaltung der Konsistenz nutzen.

Betrachtet man nun die 564 Einträge zu abstrakten und konkreten Eingangspaketen, stellt man fest, dass die Konsistenz zwischen $I$ und $I^*$ mit einem MAE von 0,0054 (Standardabweichung 0,0085) eingehalten wird.

Die 113 Einträge zu den abstrakten und konkreten Sendungen, welche sich in den Log-Einträgen finden, weisen einen MAE von 0,00011 (Standard Abweichung <0,0001) auf.

## 12.2 Fallstudie 2 – Flugzeuge

Sowohl bei Baohong (Baohong, 2007) als auch bei Hong (Hong, 2013) wird das Beispiel einer Flugzeugformation über feindlichem Gebiet genutzt, um die Eignung der beiden Ansätze zu demonstrieren. Dabei wird eine Formation auf zwei Abstraktionsstufen modelliert. In einer abstrakten Ebene wird die Formation mit einer einzelnen, dreidimensionalen Position beschrieben. Dies ist ausreichend, solange die Formation sich außerhalb der Radarreichweite des Feindes befindet. Bei der Simulation bewegt sich diese Formation solange über dem Gebiet, bis sie die Reichweite einer Radarstation unterschreitet. Nun wird die Formation auf die detaillierte Ebene übertragen. Hier wird sie mit der Position der einzelnen Flugzeuge modelliert und die Bewegung der einzelnen Flugzeuge simuliert. Verlassen diese schließlich die Reichweite der Radarstation, werden sie wieder zu einer einzelnen Formation zusammengefasst. Die Algorithmen $up$ und $down$ müssen die Formation in die einzelnen Flugzeuge überführen und umgekehrt. Im Folgenden wird beschrieben, wie dieses Beispiel auf eine Multi-Level-Simulation übertragen werden kann und die entsprechenden Algorithmen $up$ und $down$ gelernt werden können.

### 12.2.1 Struktur

Das grobe Systemmodell gliedert sich in drei wesentliche Entitäten: die Formation, die Radarstation und eine Luftverteidigungsanlage ($Gun$). Die Formation verfügt über ein Interface, über welches sie ihre Position an die Radarstation und die Luftverteidigungsanlage übermittelt. Diese Position ist als dreidimensionaler Vektor $p \in \mathbb{R}^3$ modelliert. Hinzukommt, wie bei Baohong, noch eine Entität für die Multi-Resolution-Funktionalitäten, der sogenannte *resolution controller* X.

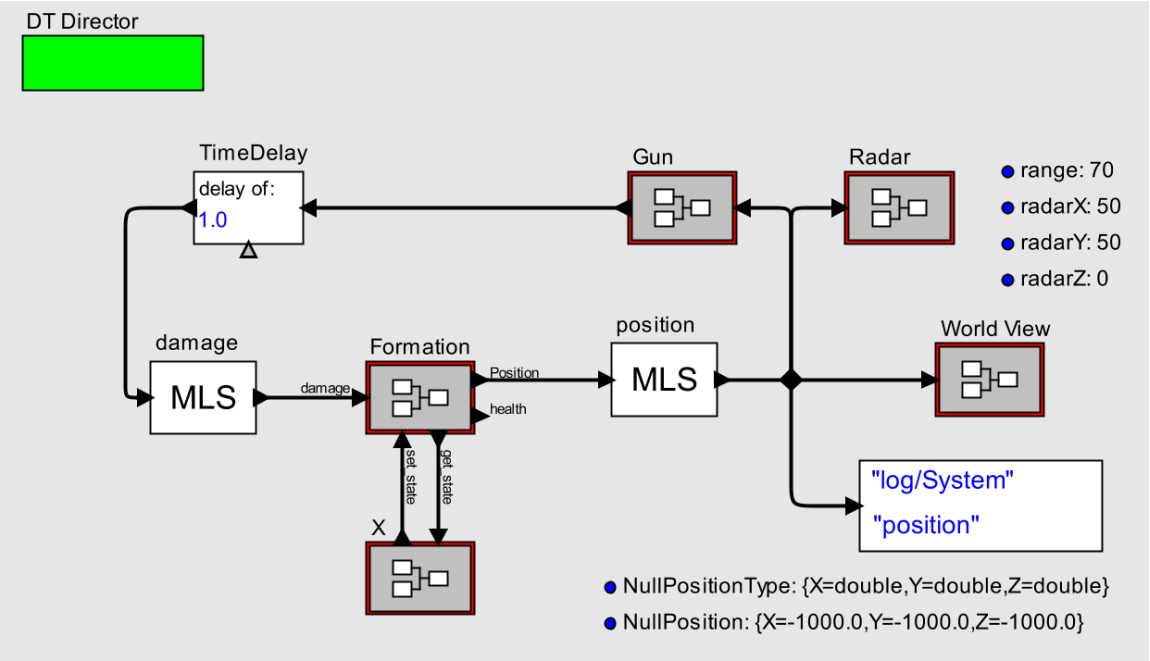



*Abbildung 12.6: Systemmodell der Fallstudie.*

Abbildung 12.6 zeigt, wie dieses Systemmodell in Ptolemy modelliert wurde. Für die *multi-resolution*-Übergänge verfügen alle Entitäten bei Baohong über Interfaces zum Lesen und Setzen des Zustandes. In diesem Beispiel betrifft dies nur die Formation. Hier kann $p$ über Interfaces manipuliert werden. Der Autor beschreibt dieses zusätzliche Interface nur für die Formation. Entsprechend werden alle weiteren Entitäten als zustandslos modelliert. Neben dem Radar wird die Position auch von der Luftverteidigungsanlage (*Gun*) gelesen. Auch diese hat eine Reichweite. Wenn die Formation in Reichweite von *Gun* ist, nimmt sie Schaden. Im Beispiel sind beide Einflusskreise disjunkt.

Der *model-resolution controller* X überwacht während der Simulation den Zustand des Modells und löst eine Änderung der Auflösung aus, wenn eine entsprechende Bedingung erfüllt wird. Im Beispiel besteht diese Bedingung daraus, dass der Abstand der Formation einen Schwellenwert unterschreitet. X wird von Baohongs Formalismus wie jede andere Entität des Modells behandelt. Entsprechend taucht X auch bei der Übertragung des Beispiels in eine Multi-Level-Simulation als Modellelement auf. Für seine Aufgabe liest X in jedem Simulationsschritt den Zustand der Formation. X verfügt in der Umsetzung innerhalb der vorliegenden Arbeit über zwei separate Zustandsvariablen, $R\_State$ und $W\_State$. $R\_State$ wird auf den aktuellen Zustand der Formation gesetzt, wenn die Bedingung von *Falsch* zu *Wahr* wechselt. Immer, wenn $W\_State$ einen von *null* verschiedenen Wert hat, überschreibt X den Zustand der Formation mit diesem Wert. $W\_State$ wird allerdings nie innerhalb des Grobmodells verändert. Dies geschieht nur im Rahmen einer Multi-Level-Simulation. Es ist

anzumerken, dass dieses Systemmodell auch allein lauffähig ist. Hierzu muss lediglich die Bedingung auf die Konstante *Falsch* gesetzt werden.

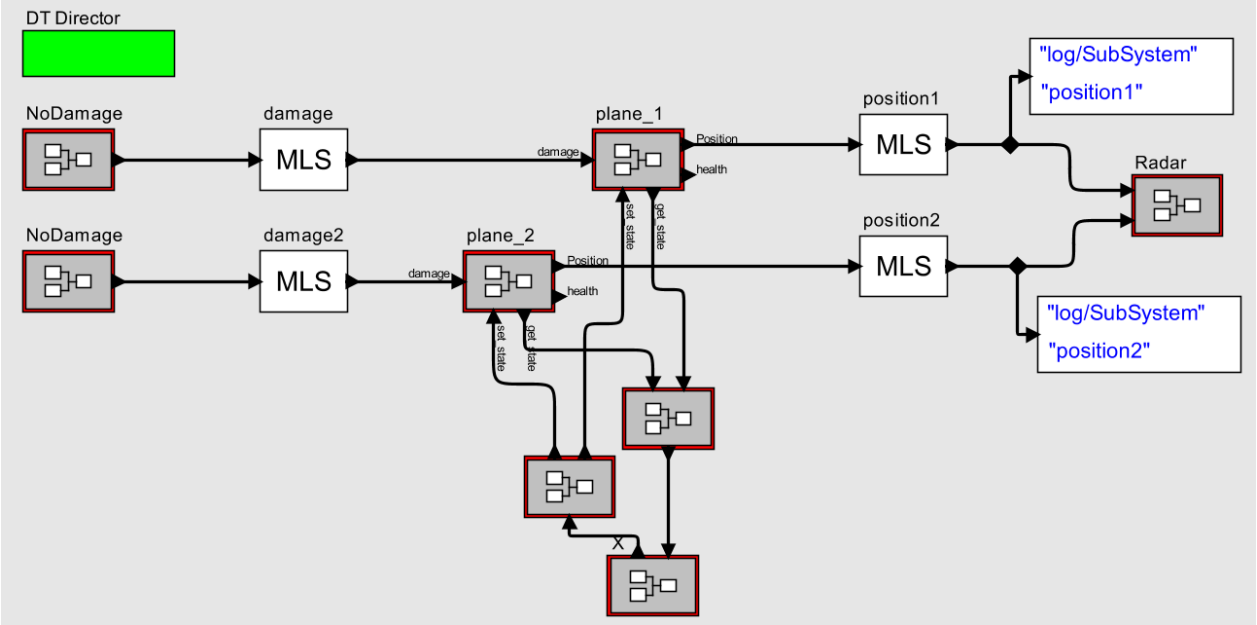



*Abbildung 12.7: Teilsystemmodell der Fallstudie.*

Wenn die Formation in Reichweite des Radars gelangt, aktiviert X bei Baohong ein detailliertes Modell der Formation mit mehreren Einzelpositionen sowie ein detailliertes Radarmodell. Die Schnittstelle zwischen beiden Entitäten überträgt die detaillierten Positionen der einzelnen Flugzeuge der Formation. In der hier vorliegenden Fallstudie setzt sich die Formation aus zwei Flugzeugen zusammen. Entsprechend ist der Zustand dieser detaillierten Formation $p^* \in \mathbb{R}^3 \times \mathbb{R}^3$ durch zwei dreidimensionale Positionen definiert.

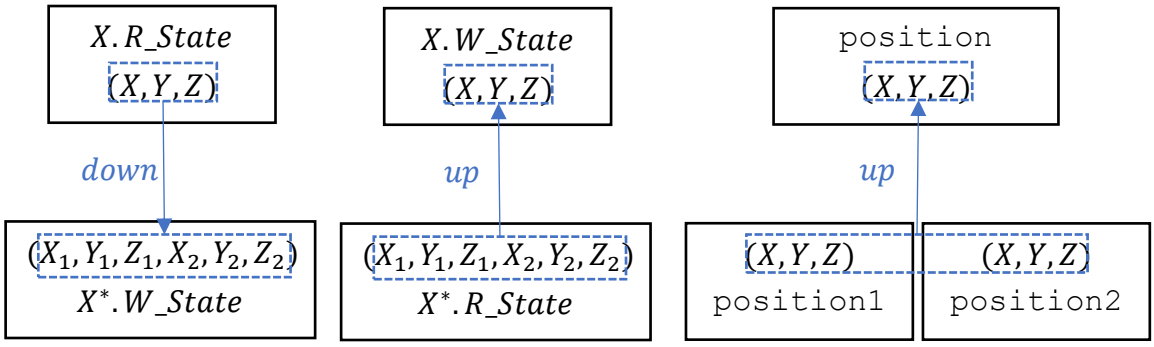



*Abbildung 12.8: Zusammenhang zwischen den MLS-Variablen beider Ebenen.*

Bei einer Multi-Level-Simulation werden diese detaillierten Entitäten in einem detaillierten Teilsystemmodell zusammengefasst. Abbildung 12.7 zeigt dieses Modell, wie es für die

Fallstudie in Ptolemy implementiert wurde. Hierbei werden die beiden Flugzeuge als separate Aktoren modelliert. Dies hat große Ähnlichkeit mit der Umsetzung bei Hong (Hong, 2013). Beide übermitteln ihre Position separat an das Radar. Zudem verfügt auch das Teilsystemmodell über einen *resolution controller* X*. Dieser funktioniert analog zu seinem Spiegelbild im Detailmodell. Auch hier wird der Zustand der Formation, bestehend aus der Position der beiden Flugzeuge, überwacht. Zentraler Unterschied ist, dass dieser seinen $R\_State$ schreibt, wenn die Flugzeuge den Sichtbereich des Radars wieder verlassen. Auch das detaillierte Modell kann isoliert simuliert werden.

Abbildung 12.8 gibt einen Überblick über die MLS-Variablen beider Modelle und wie diese durch $up$ und $down$ verbunden sind. Zunächst sind die beiden *resolution controller* durch ihre Variablen miteinander verbunden. So bildet $down$ den groben $X.R\_State$ auf den detaillierten $X^*.W\_State$ ab. Auf der anderen Seite bildet $up$ den detaillierten $R\_State$ auf den groben $W\_State$ ab.

Während einer Multi-Level-Simulation wird mit den beiden Variablen der Wechsel des Abstraktionsgrades realisiert. Zunächst ist nur die Grobsimulation „aktiv". Wenn die Formation den Sichtbereich der Radarstation betritt, schreibt $X$ seine Variable $R\_State$. Diese wird durch $down$ auf die Variabe $W\_State$ des detailierten Controllers $X^*$ abgebildet. Der Controller setzt nun den Zustand der beiden Flugzeuge. Hierdurch ist die Detailsimulation „aktiviert". Sie simuliert die Bewegung der beiden Flugzeuge durch den Sichtbereich der Radarstation. Verlassen die Flugzeuge den Sichtbereich der Radarstation wieder, schreibt $X^*$ seine Variable $R\_State$.

Diese wird durch $up$ auf ihr grobes Gegenstück $X.W\_State$ abgebildet. Abschließend setzt $X$ den Zustand der Formation auf diesen neuen Wert. Die Detailsimulation ist nun nicht mehr aktiv. Die MLS-Plattform wurde dahingehend angepasst, dass sie registriert, ob eine Detailsimulation „aktiv" ist. Hierfür werden die beiden $R\_State$-Variablen überwacht. Nur wenn die Detailsimulation notwendig ist, führt auch das Detailmodell einen Simulationsschritt aus. Diese Anpassung ermöglicht es, unnötige Berechnungen der Detailsimulation zu vermeiden.

Zudem werden in jedem Simulationsschritt, in dem die Detailsimulation aktiv ist, die detaillierten Positionen der beiden Flugzeuge (*position1* und *position2*) auf die grobe Variable *position*, abgebildet. Hier kommt erneut $up$ zum Einsatz. Dadurch kann auch die grobe *Gun* weiterhin die Position der Formation über ihre grobe Schnittstelle lesen.

Zusammen genommen entsteht der dynamische Wechsel zwischen den Abstraktionsgraden, wie er in den Arbeiten von Baohong (Baohong, 2007) und Hong (Hong, 2013) beschrieben

wird. Im nächsten Schritt werden geeignete $up$- und $down$-Algorithmen für die Fallstudie ermittelt.

### 12.2.2 Bestimmung von *up* und *down*

Auch bei dieser Fallstudie wurde $MAE_{max}$ mit 0,01 gewählt. Zunächst wurde die Detailsimulation mehrfach mit variierten Startpositionen der Flugzeuge durchgeführt. So wurden insgesamt 10000 detaillierte Flugzeugpositionen als Trainingsmenge erzeugt.

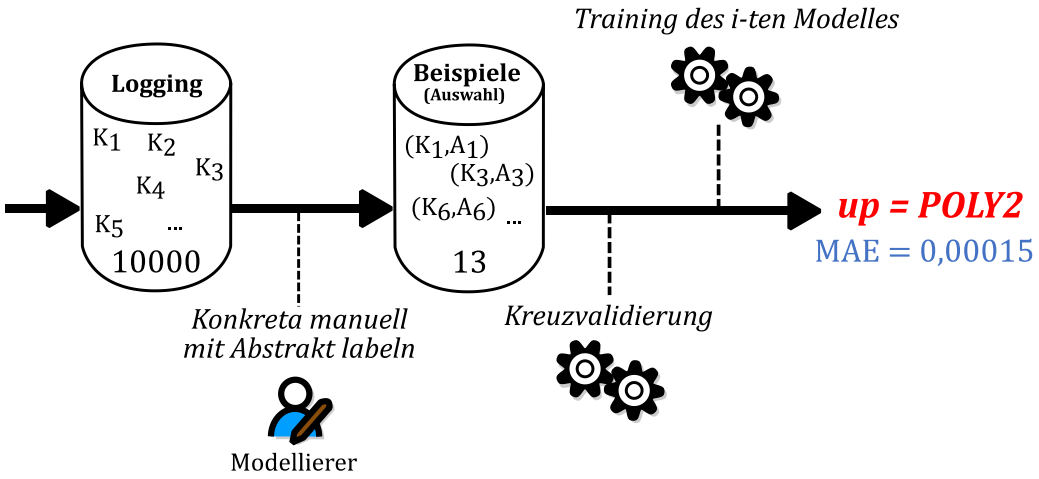



*Abbildung 12.9: Bestimmung von up für das Flugzeug-Szenario.*

Es wurden zunächst 10 der 10000 Positionsbeispiele manuell mit Formationspositionen gelabelt. Bei der Kreuzvalidierung liefert POLY2 den geringsten Fehler (0,506). Trainiert mit allen 10 Beispielen wurde ein MAE von 0,0158 ermittelt. Da dieser leicht über $MAE_{Max}$ liegt, wurden drei weitere Beispiele gelabelt. Erneut lieferte die Kreuzvalidierung POLY2 als besten Kandidaten (0,020). Mit allen 13 Beispielen kann hier ein MAE von 0,00015 erreicht werden. Dies unterschreitet $MAE_{max}$. Entsprechend wird das trainierte Modell als $up_{formation}$ verwendet.

Abbildung 12.9 fasst diesen Ablauf kurz zusammen. In Tabelle 12.2 findet sich ein Überblick über die beiden Durchläufe während der Kreuzvalidierung.

| **Anzahl der Beispiele** | POLY2 CV | POLY3 CV | POLY4 CV | POLY5 CV | NN-10x1 CV | NN-200x5 CV | **Final Training** |
|---|---|---|---|---|---|---|---|
| **10** | 0,506 | 1,951 | 3,740 | 6,250 | 11,946 | 8,057 | **POLY2 0,0158** |
| **13** | 0,020 | 0,691 | 1,679 | 2,880 | 13,088 | 5,202 | **POLY2 0,00015** |

*Tabelle 12.2: Zusammenfassung des Lernvorgangs für up.*

Mit Hilfe von $up_{fomation}$ wird nun das automatische Labeling durchgeführt. Dabei wird der in Kapitel 10 dargestellt Prozess durchlaufen. Der resultierende Ablauf wird in Abbildung 12.10 zusammengefasst. Zunächst werden aus den 10000 detaillierten Beispielen für die Position der Flugzeuge zufällig 1000 ausgewählt und mit $up_{fomation}$ gelabelt. Diese Beispielmenge aus Flugzeugpositionen mit zugehörigen Formationspositionen weist einen MAE von 0,000135 auf.

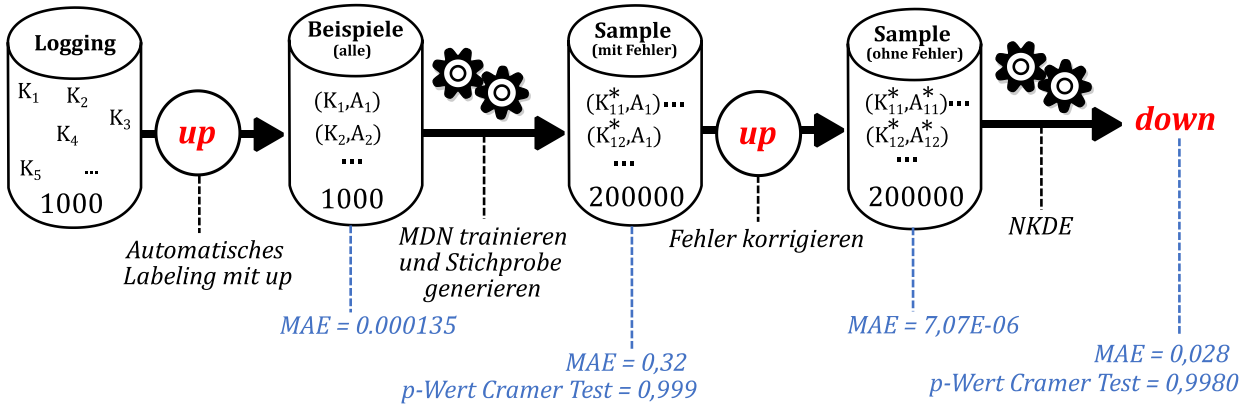



*Abbildung 12.10: Bestimmung von down für das Flugzeug-Szenario.*

Mit diesen 1000 gelabelten Beispielen wird ein MDN trainiert. Um eine große Trainingsmenge für NKDE zu erhalten, wird das trainierte MDN 200-mal auf die 1000 Konkreta aus der Beispielmenge angewandt. Der Cramer-Test liefert einen p-Wert von mehr als 0,999. Die generierte Menge weist nun einen MAE von 0,32 (gegenüber dem Orakel) auf.

Der Fehler wird erneut mithilfe von $up_{fomation}$ korrigiert und ein NKDE trainiert. Dieser wird in der Fallstudie als $down$ genutzt. Er weist einen p-Wert von 0,998 gegenüber den Trainingsdaten auf. Um den MAE bewerten zu können, wurde die Grobsimulation ausgeführt und die Belegungen der Positionsvariablen der Formation zu einer Validierungsmenge (N=1000) zusammengefasst. Hier wird ein Wert von 0,028 erreicht.

### 12.2.3 Durchführung

Die vorgestellte Multi-Level-Simulation wurde über insgesamt 13000 Zeitschritte hinweg simuliert. Es wurden insgesamt 100 Überflüge simuliert. Zu Beginn jeden Überflugs wurde die Formation auf eine Startposition gesetzt. Während des Simulationslaufes wurden erneut die Ein- und Ausgaben von $up$ und $down$ aufgezeichnet. Hierbei weist $down$ gegenüber einem Orakel den MAE von 0,0028 (Standardabweichung = 0,0134) auf. Bei $up$ liegt der MAE bei 0,00186 (Standardabweichung = 0,000599).

Abbildung 12.11 zeigt den Ablauf eines der Überflüge. In der Abbildung wird der Zustand innerhalb der Grobsimulation in schwarz und der Zustand der Detailsimulation in Rot dargestellt. Sobald diese in den Sichtbereich der Radarstation eindringt, wird auch die Position der beiden Flugzeuge dargestellt. In der detaillierten Simulation reagieren die Flugzeuge auf die Radarüberwachung durch eine veränderte Formation. Sie entfernen sich voneinander. Die gestrichelte Linie zeigt die Position der Formation, wie sie während des Überflugs der Radarstation in der Grobsimulation an *Gun* weitergegeben wird. Nach dem Verlassen des Bereiches wird wieder nur die Formation der groben Ebene dargestellt.

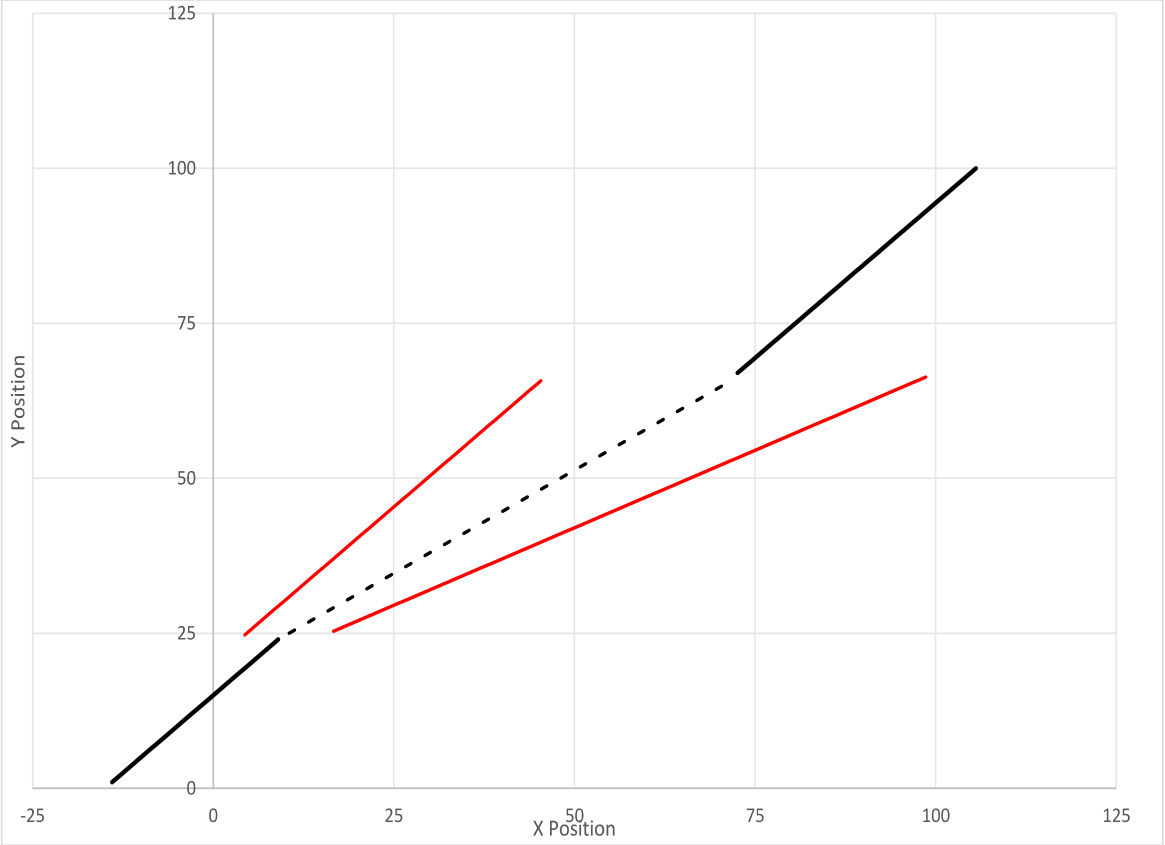



*Abbildung 12.11: Visualisierung eines einzelnen Überflugs innerhalb der Fallstudie.*

## 12.3 Zusammenfassung

In beiden Fallstudien fügt sich die Architektur mit den gelernten *up* und *down* zu einer Abstraktionsebenen-übergreifenden Simulation zusammen.

Die Ergebnisse der Cramer-Tests mit den p-Wert von 0,986 und 0,998 zeigen, dass die in Abschnitt 4.3.2 geforderte Verteilungskonformität der Konkreta auch innerhalb der Integration erreicht werden kann. Auch die in Abschnitt 4.3.1 dargestellte Konsistenzanforderung wird, bis auf den definierten Trainingsfehler, eingehalten.

Das in der zweiten Fallstudie aufgegriffene, dynamische Beispiel zeigt zudem, dass die vorgeschlagene Architektur einer Multi-Level-Simulation nicht auf statische Szenarien mit festen Kombinationen von Abstraktionsebenen beschränkt ist. Hierzu muss lediglich ein geeigneter *resolution controller* modelliert werden. Dies entspricht dem Stand der Forschung für MRE (Baohong, 2007). Selbst die Implementierung der Architektur innerhalb der Plattform musste nicht geändert werden, um das dynamische Beispiel simulieren zu können. Mit den dargestellten, marginalen Erweiterungen der Plattform ist es zudem möglich, zusätzliche Ressourceneinsparungen aus der dynamischen Natur der Fallstudie heraus zu realisieren.

# 13 Fazit

## 13.1 Zusammenfassung

Simulation ist ein weitverbreitetes Mittel, um die Entwicklung von Systemen zu unterstützen, um Erfahrungen zu vermitteln und um wissenschaftliche Erkenntnisse zu gewinnen. Auch als Unterhaltungsmedium ist sie von hoher Bedeutung. In allen diesen Bereichen hat die Wahl des Abstraktionsgrades der simulierten Modelle eine elementare Bedeutung.

Die vorliegende Arbeit ist am Beispiel des Einsatzes von Simulation zur Bewertung von experimentierfähigen Modellen im Rahmen des Entwicklungsprozesses komplexer Systeme motiviert. Hier ist es, unter den Ressourcenbeschränkungen eines Entwicklungsprojektes, notwendig, zwischen der Ganzheitlichkeit des betrachteten Systemausschnittes und dem Detailgrad, mit dem dieser Ausschnitt modelliert wird, abzuwägen.

Das Lieferkettenbeispiel zeigt, dass es möglich ist, diesem Abwägen mit einer Multi-Level-Simulation zu begegnen. Hierbei wird ein grobes Modell des gesamten Systems mit einzelnen, detaillierten Modellen von Teilsystemen, im Sinne einer Modellkopplung, verbunden. Dies birgt das Potenzial, die ganzheitliche Perspektive mit dem für bestimmte Fragestellungen notwendigen, punktuellen Detailgrad zu verbinden. Auf diese Weise können die verfügbaren Modellierungs- und Rechenressourcen effizient eingesetzt werden.

Da die Modelle auf unterschiedlichen Abstraktionsstufen angesiedelt sind, ist es häufig nicht möglich, sie direkt miteinander zu verbinden. Abstraktere Modelle werden in der Regel abstraktere Zustandsräume verwenden und entsprechend abstraktere Datentypen an den Schnittstellen nutzen. Der Zusammenhang wurde als Variablenabstraktion definiert. Sie zeichnet sich dadurch aus, dass es für einen abstrakten Zustand, bzw. für eine abstrakte Variable, eine Äquivalenzklasse von Belegungen entsprechender, konkreter Variablen geben kann. Als ein anschauliches Beispiel hierfür dienten die abstrakten Pakete, deren Volumen aus einer Vielzahl von Kombinationen aus Höhe, Breite und Tiefe gebildet werden konnten.

Im Rahmen dieser Arbeit wurden hieraus zwei Anforderungen an eine Multi-Level-Simulation abgeleitet. Die Konsistenzanforderung fordert, dass bei einem Multi-Level-Schritt die Abstraktionsbeziehung zwischen den Belegungen der entsprechenden Variablen gewahrt bleibt. Abstrakta dürfen immer nur Konkreta aus der entsprechenden Äquivalenzklasse zugeordnet werden und umgekehrt. Die Verteilungsanforderung fordert zudem, dass die Wahrscheinlichkeit der Konkreta unter der Bedingung der, durch die Grobsimulation erzeugten, Konkreta der korrekten Verteilung folgt. Dies stellt sicher, dass die Verteilung der Konkreta, welche in die Detailsimulation übergehen, plausibel bleibt.

Seit den 90er Jahren befasst sich eine Vielzahl von Ansätzen mit Schnittstellen zwischen Modellen unterschiedlicher Abstraktionsstufen und deren Kopplung. Auch der dynamische Austausch von Modellen oder Teilmodellen während eines Simulationslaufes wird wiederholt diskutiert. Von zentraler Bedeutung ist in diesen Arbeiten das starke Konsistenzkriterium von Davis *(Davis, 1998)*. Dieses erzwingt eine 1-zu-1-Beziehung zwischen Abstrakta und Konkreta und so eine Eindeutigkeit von $down$. Dies verletzt aber in fast allen Fällen die Verteilungsanforderung und führt zu unplausiblen Eingaben für die Detailsimulation. Auch bei einem naiven Adapter für das Lieferkettenbeispiel wurde diese Plausibilität verletzt.

Viele der bestehenden Ansätze sehen zudem vor, dass der Modellierer $up$ und $down$ manuell angibt. Dies birgt für die Multi-Level-Simulation das Risiko, dass der Modellierungsaufwand für eine vollständige Detailsimulation in zusätzlichen Aufwand bei der Beschreibung von $up$ und $down$ fließen muss. Aus diesem Grund wurde eine Lösung auf Basis von maschinellem Lernen gesucht, welche mit wenigen, gelabelten Beispielen in der Lage ist, $up$ und $down$ zu generieren.

Entsprechend befasst sich diese Arbeit mit der Frage:

1. **Wie kann eine Architektur für lernende Multi-Level-Simulation aussehen?**

Um diese Frage zu beantworten, wurde eine entsprechende Architektur entworfen und formal definiert. Auch die Konsistenz- und Verteilungsanforderungen, welche sich für $up$ und $down$ ableiten, wurden hier formal beschrieben.

Bei einer lernenden Multi-Level-Simulation wird in einer, der Ausführung vorgelagerten, Trainingsphase zunächst die Detailsimulation ausgeführt. Wenige der Detailvariablenbelegungen werden manuell gelabelt, um $up$ zu trainieren. Mit diesem kann nun eine größere Trainingsmenge für $down$ generiert werden.

Um beide Algorithmen generieren zu können, wurden folgende Forschungsfragen untersucht.

2. **Welche Methode des maschinellen Lernens eignet sich zur Bestimmung von $up$?**

3. **Welche Methode des maschinellen Lernens eignet sich zur Bestimmung von $down$?**

Zur Abschätzung des durch $up$ und $down$ eingeführten Fehlers in Bezug auf die Konsistenzanforderung wurde der *mean absolut error* (MEA) bzw. der Absolute Round Trip Error (ARTE) eingeführt. Um die Verteilungsanforderung bewerten zu können, wurde der multivariate, nicht parametrische Cramer-Test genutzt.

Als Antwort auf 2. wurde $up$ als Regressionsproblem formuliert. Es gibt eine Vielzahl von Verfahren mit welchen derartige Probleme gelöst werden können. Da sich diese jeweils nicht für alle Datensätze eignen, wurde ein Prozess vorgestellt, welcher mithilfe von Kreuzvalidierung nach dem besten Verfahren sucht. Zudem wurde in diesem Prozess sukzessive manuell weitere Beispiele gelabelt, bis ein gewünschter MAE unterschritten wurde. Zur Evaluation wurden auf Basis von Abstraktionsfunktionen, welche der Literatur entnommen wurden, Datensätze erzeugt und mit dem Prozess gelernt. Schließlich wurde der MAE gegenüber einer Validierungsmenge bestimmt. Häufig konnten hierbei auch sehr kleine MAE (<0,1% des Wertebereiches) mit wenigen Beispielen (<20) erreicht werden. Für einige Funktionen empfiehlt sich jedoch dennoch die manuelle Definition von $up$. Hierfür wurde eine entsprechende Abbruchbedingung im Prozess eingeführt, welche die manuelle Definition von nach $N_{max}$ gelabelten Beispielen vorsieht.

Als Antwort auf 3. wurde $down$ als Stichprobe einer bedingten Wahrscheinlichkeitsverteilung aufgefasst. Hierzu musste diese bedingte Wahrscheinlichkeitsverteilung geschätzt werden (in der Literatur: *conditional density estimation,* kurz CDE). Anschließend werden Stichproben der geschätzten Verteilung generiert. Bei den Verfahren aus der Literatur ergibt sich über ihre Parameter eine Trade-off zwischen der erreichten Abweichung der generierten Konkreta bezüglich des ARTE und der Genauigkeit, mit der die Verteilung ermittelt wird (Verteilungsanforderung). Um dies aufzulösen, wurden zwei Verfahren kombiniert. *mixture density networks* (MDN) wurden genutzt, um zunächst die Verteilung mit großer Genauigkeit zu ermitteln (p-Wert < 0,99). Hierbei ist jedoch der ARTE recht groß (ARTE > 0,7). Deshalb wurde mit dem trainierten MDN eine große Anzahl an Trainingsdaten für einen *$\epsilon$-neighbor kernel density estimator* (NKDE) generiert. Der NKDE erreicht schließlich einen sehr kleinen ARTE (<0,011) ohne die Verteilung nennenswert zu verschlechtern (p-Wert < 0,98).

Abschließend wurde folgende Forschungsfrage untersucht:

4. **Fügen sich die Lehrverfahren für die Algorithmen $up$ und $down$ tatsächlich mit der Architektur zu einer Multi-Level-Simulation zusammen?**

Hierzu wurde eine Plattform für lernende Multi-Level-Simulation implementiert. Diese setzt das formale Modell um. Durch eine Integration der Simulationsumgebung Ptolemy können Modelle angeschlossen werden. Zudem werden die ermittelten $up$ und $down$ in die Plattform mit entsprechenden Austauschformaten integriert. Auf Basis dieser Plattform wurden zwei Fallstudien durchgeführt.

Zunächst wurde die Implementierung des eingangs erwähnten Lieferkettenbeispiels als lernende Multi-Level-Simulation umgesetzt. Dabei wurde das Grobmodell der Lieferkette mit einem Detailmodell des Standortes der Kommissionierung verbunden. Die Algorithmen $up$ und

*down* wurden entsprechend der beiden Prozesse generiert. Anschließend wurde die Multi-Level-Simulation durchgeführt und die Einhaltung beider Anforderungen untersucht.

Als zweite Fallstudie wurde ein militärisches Beispiel von Baohong (Baohong, 2007) umgesetzt. Anders als die erste Fallstudie ist diese auf einen dynamischen Wechsel der Abstraktionsebenen ausgelegt. Dabei wechseln einzelne Entitäten der Simulation unter definierten Bedingungen (Trigger) die Abstraktionsebene. Dies wird im Ansatz von Baohong durch mehrere explizite Modellelemente realisiert, welche den Simulationszustand auf die Trigger prüfen. Sie wurden ebenfalls in die Implementierung der Fallstudie übernommen. Allein hierdurch war es möglich, auf Basis der Architektur und der Plattform, diesen dynamischen Wechsel umzusetzen. Erneut wurde die Multi-Level-Simulation durchgeführt und die Einhaltung beider Anforderungen untersucht.

In beiden Fallstudien konnten die Anforderungen eingehalten werden.

## 13.2 Diskussion und Ausblick

Die vorliegende Arbeit bricht als eine der ersten das starre Konzept der starken Konsistenz von Davis *(Davis, 1998)* auf. Sie erlaubt der Multi-Level-Simulation, durch das nicht-determinierte *down*, indeterministisches Verhalten. Dabei ist der Indeterminismus keinesfalls mit Beliebigkeit gleichzusetzen. Der Zufall bleibt durch die CDE im Rahmen einer plausiblen, durch Daten gestützten, Verteilung. Der hybride CDE-Ansatz sorgt dafür, dass die erzeugten Konkreta im Sinne der Konsistenz zu den ursprünglichen Abstrakta passen. Gleichzeitig liefert die vorliegende Arbeit Instrumenten zur Bewertung des hierdurch eingebrachten Fehlers. Diese neue Perspektive stellt eine essenzielle Erweiterung des Standes der Forschung dar. Die dargestellte Architektur macht diesen verteilungskonformen Indeterminismus erst möglich.

Die Architektur wurde für den Anwendungsfall einer statischen Kopplung von Simulationen entworfen. Entsprechend bietet sie keine explizite Infrastruktur für den dynamischen Anwendungsfall. Diese Funktionalität konnte in der zweiten Fallstudie allein auf der Ebene des Modells umgesetzt werden. Dies ist ein Beleg für die Flexibilität der gefunden Architektur.

Dennoch liegt ein offensichtlicher Anknüpfungspunkt für weitere Arbeiten in der Erweiterung der Architektur um derartige explizite Mechanismen. So wäre es interessant, den *resolution controller* nicht als Modellelement abzubilden, sondern das dahinterliegende Trigger-Konzept explizit in die Architektur aufzunehmen. In der aktuellen Architektur setzt sich ein Multi-Level-Modell statisch aus einem abstrakten und einem konkreten Modell zusammen. Für den dynamischen Fall ist eine Repräsentation dafür, ob das konkrete Modell aktiv ist, notwendig.

Diese könnte beispielsweise als Funktion des Zustands des abstrakten Modells wie folgt definiert werden:

$$aktiv: \bar{S} \rightarrow \{true, false\}$$

Der Multi-Level-Schritt müsste diese Bedingung berücksichtigen und bei Aktivwerden des konkreten Modells dieses initiierten. Auch die MLS-Plattform kann um entsprechende Funktionalitäten erweitert werden.

Die Plattform erlaubt auch die verteilte Simulation. Hierbei wird jedes Modell auf einem eigenen Rechenknoten ausgeführt. Auch die Umsetzung dieser verteilten Multi-Level-Simulation in Cloud-Infrastrukturen ist möglich (Erbel, 2020). Das DT-Zeitmodell dient als Basis für die Architektur. Hiermit kann bei hinreichend kleiner Schrittweite beliebig gut jedes andere Zeitmodell abgebildet werden. Dies ist im Rahmen der hier durchgeführten Untersuchungen, die sich vor allem auf $up$ und $down$ fokussieren, ausreichend. Sehr kleine Schrittweiten führen in einer verteilten Infrastruktur allerdings zu einem steigenden Kommunikationsbedarf zwischen den Rechenknoten. Im DT-Zeitmodell müssen auch nicht veränderte Variablenbelegungen in jedem Schritt kommuniziert werden. Dies kann technisch gelöst werden, in dem nur Änderungen übertragen werden. Trotzdem ist auch eine Erweiterung des Zeitmodells um ein ereignisbasiertes Konzept interessant. Hierbei würde bereits auf konzeptioneller Ebene nur dann kommuniziert werden, wenn dies erforderlich ist.

Die Implementierung des hybriden CDE wurde insbesondere mit Blick auf eine Schonung der Rechenressourcen ausgelegt, um den Performancegewinn der Multi-Level-Simulation nicht zu gefährden. Zu diesem Zweck werden mit dem MDN möglichst viele Trainingspunkte generiert. Sie folgen der Verteilung der Abstrakta, welche durch $up$ den ursprünglichen Inputs zugeordnet werden. Danach ist das MDN nicht mehr erforderlich. Der NKDE speichert alle diese Punkte und erzeugt Konkreta durch Anfragen an einen Datenspeicher. Da die Anfrage mithilfe von kd-Tree-Indizierung optimiert wird, ist dies in der Praxis mit sehr kurzen Ausführungszeiten von $down$ verbunden. Allerdings müssen permanent alle generierten Punkte gespeichert bleiben. Es ist jedoch nicht notwendigerweise so, dass tatsächlich alle diese Punkte überhaupt erforderlich sind. Möglicherweise wird ein konkreter Simulationsdurchlauf z.B. nur sehr große Pakete betrachten. Eine mögliche Erweiterung liegt darin, das MDN zu speichern und nur bei konkreten Anfragen die Punkte in der Nähe der Anfragestelle zu generieren. Vergleichbar mit dynamischer Programmierung könnten dann nur diese Punkte gespeichert werden. Hierdurch könnte eine speichereffizientere Variante entwickelt werden.

In den vorliegenden Untersuchungen wurde stets von realwertigen Variablen und Vektoren ausgegangen. Andere Datentypen, wie Wahrheitswerte oder Integer, wurden nicht explizit betrachtet. Sie könnten jedoch leicht verwendet werden, indem sie auf die realen Zahlen abgebildet werden. Durch Runden könnten auch die für $up$ und $down$ verwendeten

Lernverfahren genutzt werden. Allerdings ist davon auszugehen, dass die Genauigkeit der Vorhersagen mit explizit auf diskrete Datentypen ausgelegten Verfahren, wie z.B. Clustering, gesteigert werden könnte. Dies zu untersuchen stellte einen weiteren Anknüpfungspunkt dar.

Abstraktionsbeziehungen, wie in dieser Arbeit diskutiert, kommen keinesfalls ausschließlich bei der Multi-Level-Simulation vor. Eine mögliche Anwendung liegt in dem Generieren von Testfällen aus Spezifikationen. Spezifikationen werden meist mit abstrakteren Datentypen beschrieben als jene, mit denen später die Systeme arbeiten. In der Spezifikation eines autonomen Fahrzeugs könnte von einem Abstand zum vorausfahrenden Fahrzeug die Rede sein. Im fertigen System wird dann eine zweidimensionale Position erfasst. Nun muss ein Testfall für die Implementierung manuell programmiert werden. Zwischen den Datentypen der Spezifikation und dem Testfall besteht eine Abstraktionsbeziehung (vgl. (Mauritz, 2016)). Daher ist es unter Umständen möglich, $down$ einzusetzen, um diese Testfälle zu generieren. Die Anwendung der hybriden CDE auf die Testdomäne stellt eine weitere, mögliche Richtung für zukünftige Arbeiten dar.

Ein weiteres Beispiel für die Invertierung einer Abstraktionsbeziehung ist die sogenannte *energy disaggregation*. Hierbei soll aus dem Stromverbrauch, z.B. eines Hausanschlusses, der Verbrauch einzelner Etagen ermittelt werden. Die Arbeit von Batra beschäftigt sich beispielsweise mit diesem Problem (Batra, 2014). Die von Batra untersuchten Daten wurden teilweise veröffentlicht.

Offensichtlich entspricht der Stromverbrauch des Hausanschlusses der Summe der Etagenwerte. Somit ist $down$ prinzipiell für die Aufgabe geeignet. In einem ersten Versuch wurden $down$ mithilfe der hybriden CDE auf Basis der öffentlich zugänglichen Daten trainiert und die Qualität der Disaggregation mit den für dieses Problem gängigen Ansätzen verglichen. Diese Verfahren nutzen z.B. das Frequenzband der Strom- und Spannungswerte, um einzelnen Verbräuche zu identifizieren und den Etagen zuzuordnen.

Erste, noch unveröffentlichte, Ergebnisse dieser Untersuchungen sind vielversprechend. Es konnte ein ARTE von 3,91 erreicht werden. Die Verbrauchswerte des Hausanschlusses haben einen Mittelwert von 23363 kWh. Somit entspricht der ARTE einem Fehler von 0,016%. Dies ist die Abweichung der Summe der Prognosen der Etagenverbräuche zu dem tatsächlichen Verbrauch des Hausanschlusses. Ebenso relevant ist die Frage, ob die einzelnen Etagenverbräuche durch $down$ getroffen werden. Hierfür wird in der Veröffentlichung von Batra der *normalized error in assigned power* (NEP) genutzt. Dieser liegt in den Voruntersuchungen mit einem durchschnittlichen Wert von 0,212 deutlich unter dem Durchschnitt der bestehenden Verfahren. Diese haben bei Batra einen durchschnittlichen NEP von 1,04 erreicht. Hierbei handelt es sich jedoch nur um erste Ad-hoc-Ergebnisse. Auch sie motivieren jedoch tiefergehende Untersuchungen.

# Literatur


Ackley, D.H., Hinton, G.E., Sejnowski, T.J., 1985. A Learning Algorithm for Boltzmann Machines*. Cognitive Science 9, 147–169. https://doi.org/10.1016/S0364-0213(85)80012-4

Ambrogioni, L., Güçlü, U., van Gerven, M.A.J., Maris, E., 2017. The Kernel Mixture Network: A Nonparametric Method for Conditional Density Estimation of Continuous Random Variables. https://arxiv.org/abs/1705.07111.

Baohong, L., 2007. A Formal Description Specification for Multi-Resolution Modeling Based on DEVS Formalism and its Applications. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 4, 229–251. https://doi.org/10.1177/154851290700400302

Baohong, L., Huang, K. Di, 2005. Research on consistency in multi-resolution model family. Journal of System Simulation 17, 2057–2060. https://doi.org/10.16182/j.cnki.joss.2005.09.004

Baringhaus, L., Franz, C., 2004. On a new multivariate two-sample test. Journal of Multivariate Analysis 88, 190–206. https://doi.org/10.1016/S0047-259X(03)00079-4

Batra, N., Parson, O., Berges, M., Singh, A., Rogers, A., 2014. A comparison of non-intrusive load monitoring methods for commercial and residential buildings, https://arxiv.org/abs/1408.6595.

Bentley, J.L., Louis, J., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM 18, 509–517. https://doi.org/10.1145/361002.361007

Bishop, C.M., 2013. Mixture Density Networks. Journal of Chemical Information and Modeling 53, 1689–1699. https://doi.org/10.1017/CBO9781107415324.004

Bishop, C.M., 2006. Pattern Recognition and Machine Learning, Journal of Electronic Imaging, Information science and statistics. Springer, New York. https://doi.org/10.1017/CBO9781107415324.004

Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A., 2012. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. Proceedings of the 9th International MODELICA Conference 173–184. https://doi.org/10.3384/ecp12076173

Broy, M., Stolen, K., 1994. Specification and Refinement of Finite Dataflow Networks - a Relational Approach, in: Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT). Springer Berlin Heidelberg, Berlin, Heidelberg, S. 247–267.

Camacho, E.F., Bordons, C., 2007. Model Predictive control, Advanced Textbooks in Control and Signal Processing. Springer-Verlag, London. https://doi.org/10.1007/978-0-85729-398-5

Cellier, F.E., 1991. Continuous system modeling. Springer Science & Business Media, New York.

Claes, R., Holvoet, T., 2009. Multi-model traffic microsimulations, in: Proceedings - Winter Simulation Conference, WSC '09. Winter Simulation Conference, Austin, Texas, S. 1113–1123. https://doi.org/10.1109/WSC.2009.5429657

Dangelmaier, W., Mueck, B., 2004. Using Dynamic Multiresolution Modelling to Analyze Large Material Flow Systems, in: Proceedings of the 2004 Winter Simulation Conference, 2004., WSC '04. Winter Simulation Conference, Washington, D.C., S. 645–652. https://doi.org/10.1109/WSC.2004.1371522

Davis, P.K., 1995. An Introduction to Variable-Resolution Modeling. Naval Research Logistics (NRL) 42, 151–181.

Davis, P.K., 1992. An Introduction to Variable-Resolution Modeling and Cross-Resolution Model Connection, in: Hillestad, P.K.D. and R. (Hrsg.), Proceedings of the Conference on Variable-Resolution Modeling.

Davis, P.K., Bigelow, J.H., 1998. Experiments in Multiresolution Modeling (MRM). CA: RAND Corporation, Santa Monica.

Davis, P.K., Bigelow, J.H., McEver, J., 2000. Effects of Terrain, Maneuver Tactics, and C41sr on the Effectiveness of Long Range Precision Fires: A Stochastic Multiresolution Model (Pem) Calibrated to High-Resolution Simulation. RAND Corporation.

Davis, P.K., Huber, R.K., 1992. Variable-Resolution Combat Modeling, in: Proceedings of the Conference on Variable-Resolution Modeling. RAND Corporation, Santa Monica.

Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y., 2003. Taming heterogeneity - The ptolemy approach. Proceedings of the IEEE 91, 127–143. https://doi.org/10.1109/JPROC.2002.805829

ELKI, o. J. Environment for Developing KDD-Applications Supported by Index-Structures [WWW Document]. URL https://elki-project.github.io/ (zugegriffen 10.29.20).

Engel, P., Meise, S., Rausch, A., Tegethoff, W., 2019. Modeling of Automotive HVAC Systems Using Long Short-Term Memory Networks, in: ADAPTIVE 2019: The Eleventh International Conference on Adaptive and Self-Adaptive Systems and Applications. S. 48–55.

Erbel, J., Wittek, S., Grabowski, J., Rausch, A., 2020. Dynamic Management of Multi-level-simulation Workflows in the Cloud, in: Communications in Computer and Information Science. Springer, S. 21–38. https://doi.org/10.1007/978-3-030-45718-1_2

FMI, o. J. Functional Mock-up Interface Specification 2.0.1 [WWW Document]. URL https://github.com/modelica/fmi-standard/releases/download/v2.0.1/FMI-Specification-2.0.1.pdf (zugegriffen 9.3.20).

Frantz, F.K., 1995. A taxonomy of model abstraction techniques, in: Winter Simulation Conference Proceedings, 1995., WSC '95. IEEE Computer Society, Washington, DC, USA, S. 1413–1420. https://doi.org/10.1109/WSC.1995.479055

Georgii, H.-O., 2007. Stochastik, Stochastik. De Gruyter. https://doi.org/10.1515/9783110206777

Goodman, L.A., 1960. On the Exact Variance of Products. Journal of the American Statistical Association 55, 708–713. https://doi.org/10.1080/01621459.1960.10483369

Gutlein, M., German, R., Djanatliev, A., 2018. Towards a Hybrid Co-Simulation Framework: HLA-Based Coupling of MATSim and SUMO, in: 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT). IEEE, S. 1–9. https://doi.org/10.1109/DISTRA.2018.8601004

Hong, S.-Y., Kim, T.G., 2013. Specification of multi-resolution modeling space for multi-resolution system simulation. SIMULATION 89, 28–40. https://doi.org/10.1177/0037549712450361

Huber, D., Dangelmaier, W., 2011. A method for simulation state mapping between discrete event material flow models of different level of detail, in: Proceedings - Winter Simulation Conference, WSC '11. Winter Simulation Conference, Phoenix, Arizona, S. 2872–2881. https://doi.org/10.1109/WSC.2011.6147990

Huber, D., Dangelmaier, W., 2009. Controlled simplification of material flow simulation models, in: Proceedings - Winter Simulation Conference, WSC '09. Winter Simulation Conference, Austin, Texas, S. 839–850. https://doi.org/10.1109/WSC.2009.5429707

Isermann, R., Schaffnit, J., Sinsel, S., 1999. Hardware-in-the-loop simulation for the design and testing of engine-control systems. Control Engineering Practice 7, 643–653. https://doi.org/10.1016/S0967-0661(98)00205-6

Java, 2020. The Java® Language Specification [WWW Document]. URL https://docs.oracle.com/javase/specs/jls/se14/html/index.html (zugegriffen 10.17.20).

JPMML-Evaluator - Java Evaluator API for PMML [WWW Document], o. J. URL https://github.com/jpmml/jpmml-evaluator (zugegriffen 2.19.20).

Jun, J.B., Jacobson, S.H., Swisher, J.R., 1999. Application of discrete-event simulation in health care clinics: A survey. Journal of the Operational Research Society 50, 109–123. https://doi.org/10.1057/palgrave.jors.2600669

Kidwell, D.A. (Hrsg.), 1997. Lazy Learning, 1. ed. Springer Netherlands, Dordrecht. https://doi.org/10.1007/978-94-017-2053-3

Lamport, L., 1989. A simple approach to specifying concurrent systems. Communications of the ACM. https://doi.org/10.1145/63238.63240

Lee, E.A., 2008. Cyber physical systems: Design challenges, Proceedings - 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008. https://doi.org/10.1109/ISORC.2008.25

Lee, E.A., Neuendorffer, S., 2000. MoML - A modeling markup language in XML - Version 0.4. University of California at Berkeley, Tech. Rep.

Lückel, J., Koch, T., Schmitz, J., 2000. Mechatronik als integrative Basis flir innovative Produkte, in: VDI-Tagung: Mechatronik - Mechanisch/Elektrische Antriebstechnik. VDI-Vedag, Wiesloch.

Maier, M.W., 1998. Architecting principles for systems-of-systems. Systems Engineering 1, 267–284. https://doi.org/10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D

Mallet, D.G., De Pillis, L.G., 2007. A cellular automata model of tumor–immune system interactions. Journal of Nuclear Cardiology 14, S97–S97. https://doi.org/10.1016/j.nuclcard.2007.06.010

Mauritz, M., Howar, F., Rausch, A., 2016. Assuring the safety of advanced driver assistance systems through a combination of simulation and runtime monitoring, in: Lecture Notes in Computer Science. Springer Verlag, S. 672–687. https://doi.org/10.1007/978-3-319-47169-3_52

Modelica, o. J. The Modelica Associatio [WWW Document]. URL https://www.modelica.org/ (zugegriffen 9.3.20).

Nagel, K., Schreckenberg, M., 1992. A cellular automaton model for freeway traffic. Journal de Physique I 2, 2221–2229. https://doi.org/10.1051/jp1:1992277

Natrajan, A., 2000. Consistency Maintenance in Concurrent Representations [Dissertation]. University of Virginia.

Natrajan, A., Reynolds, Jr., P.F., Srinivasan, S., 1995. Consistency Maintenance using UNIFY. University of Virginia, Charlottesville, VA 44, 0–12.

Navarro, L., Corruble, V., Flacher, F., Zucker, J.-D., 2013. A Flexible Approach to Multi-Level Agent-Based Simulation with the Mesoscopic Representation, in: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, AAMAS '13. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, S. 159–166.

Navarro, L., Flacher, F., Corruble, V., 2011. Dynamic Level of Detail for Large Scale Agent-Based Urban Simulations, in: AAMAS '11: The 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '11. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, S. 701–708.

Neal, R.M., 1992. Connectionist learning of belief networks. Artificial Intelligence 56, 71–113. https://doi.org/10.1016/0004-3702(92)90065-6

Ören, T., 2011. A Basis for a Modeling and Simulation Body of Knowledge Index : Professionalism , Stakeholders , Big Picture , and Other BoKs. Ören, T. (2011). A Basis for a Modeling and Simulation Body of Knowledge Index : Professionalism , Stakeholders , Big Picture , and Other BoKs, 1, 40–48. 1, 40–48.

PMML, o. J. Predictive Model Markup Language - PMML 4.4.1 [WWW Document]. URL http://dmg.org/pmml/v4-4-1/GeneralStructure.html (zugegriffen 10.29.20).

Ptolemy, o. J. Generator, Regulator, Protector [WWW Document]. URL https://ptolemy.berkeley.edu/books/Systems/models/doc/books/systems/intro/GeneratorRegulatorProtector/index.html (zugegriffen 9.3.20).

Rabelo, L., Kiyoul Kim, Tae Woong Park, Pastrana, J., Marin, M., Lee, G., Nagadi, K., Ibrahim, B., Gutierrez, E., 2015. Multi resolution modeling, in: 2015 Winter Simulation Conference (WSC). IEEE, S. 2523–2534. https://doi.org/10.1109/WSC.2015.7408362

Rao, D., Wilsey, P., 2000. Dynamic component substitution in Web-based simulation, in: 2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165), WSC '00. Society for Computer Simulation International, San Diego, CA, USA, S. 1840–1848. https://doi.org/10.1109/WSC.2000.899177

Rao, D.M., 2003. Study of Dynamic Component Substitutions [Dissertation]. University of Cincinnati, OhioLINK Electronic Theses and Dissertations Center.

Rao, D.M., Wilsey, P.A., 2008. Predicting Performance Impacts due to Resolution Changes in Parallel Simulations. SIMULATION 84, 535–555. https://doi.org/10.1177/0037549708096134

Rao, D.M., Wilsey, P.A., 2006. Accelerating ATM Simulations Using Dynamic Component Substitution (DCS). Simulation 82, 235–253. https://doi.org/10.1177/0037549706067271

Rao, D.M., Wilsey, P.A., 2002. Web-based Simulation 2: Performance Prediction of Dynamic Component Substitutions, in: WSC '02. Winter Simulation Conference, San Diego, California, S. 816–824.

Rasmussen, C.E., Williams, C.K.I., 2006. Gaussian processes for machine learning. MIT Press.

Rausch, A., 2001. Componentware - Methodik des evolutionären Architekturentwurfs. Herbert Utz Verlag, München.

Reynolds, P.F., Natrajan, A., Srinivasan, S., 1997. Consistency maintenance in multiresolution simulation. ACM Transactions on Modeling and Computer Simulation (TOMACS) 7, 368–392. https://doi.org/10.1145/259207.259235

Richter, S., 2019. Statistisches und maschinelles Lernen, Statistisches und maschinelles Lernen. https://doi.org/10.1007/978-3-662-59354-7

Rothfuss, J., Ferreira, F., Walther, S., Ulrich, M., 2019. Conditional Density Estimation with Neural Networks: Best Practices and Benchmarks. https://arxiv.org/abs/1903.00954v2.

Sato, R., Praehofer, H., 1997. A discrete event system model of business system-a systems theoretic foundation for information systems analysis. I. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 27, 1–10. https://doi.org/10.1109/3468.553213

Schütte, S., 2013. Simulation Model Composition for the Large-Scale Analysis of Smart grid Control Mechanisms. Simulation Model Composition for the Large-Scale Analysis of Smart grid Control Mechanisms 14–35.

Schütte, S., Scherfke, S., Tröschel, M., 2011. Mosaik: A framework for modular simulation of active components in Smart Grids, in: 2011 IEEE 1st International Workshop on Smart Grid Modeling and Simulation, SGMS 2011. IEEE, S. 55–60. https://doi.org/10.1109/SGMS.2011.6089027

Sewall, J., Wilkie, D., Lin, M.C., 2011. Interactive Hybrid Simulation of Large-Scale Traffic. ACM Trans. Graph 30, 11. https://doi.org/10.1145/2024156.2024169

Silverman, B.W., 1982. On the Estimation of a Probability Density Function by the Maximum Penalized Likelihood Method. The Annals of Statistics 10, 795–810. https://doi.org/10.1214/aos/1176345872

Simulink, o. J. Simulink - Simulation und Model-Based Design - MATLAB & Simulink [WWW Document]. URL https://de.mathworks.com/products/simulink.html (zugegriffen 9.3.20).

SkLearn2PMML, o. J. Python library for converting Scikit-Learn pipelines to PMML.

Sugiyama, M., Takeuchi, I., Suzuki, T., Kanamori, T., Hachiya, H., Okanohara, D., 2010. Conditional Density Estimation via Least-Squares Density Ratio Estimation.

Tako, A.A., Robinson, S., 2012. The application of discrete event simulation and system dynamics in the logistics and supply chain context. Decision Support Systems 52, 802–815. https://doi.org/10.1016/j.dss.2011.11.015

Tang, Y., Salakhutdinov, R.R., 2013. Learning Stochastic Feedforward Neural Networks, in: C.J.C. Burges, L. Bottou, M.W. (Hrsg.), Advances in Neural Information Processing Systems 26. Curran Associates, Inc., S. 530–538.

Tansey, W., Pichotta, K., Scott, J.G., 2016. Better Conditional Density Estimation for Neural Networks. https://arxiv.org/abs/1606.02321v1.

Tipping, M.E., 2001. Sparse Bayesian Learning and the Relevance Vector Machine. Journal of Machine Learning Research 1, 211–244. https://doi.org/10.1162/15324430152748236

Tripakis, S., Stergiou, C., Shaver, C., Lee, E.A., 2013. A modular formal semantics for Ptolemy. Mathematical Structures in Computer Science 23, 834–881. https://doi.org/10.1017/S0960129512000278

Vapnik, V.N., 1995. The Nature of Statistical Learning Theory, The Nature of Statistical Learning Theory. https://doi.org/10.1007/978-1-4757-2440-0

VDI, 2008. Richtlinie 3633-1: Simulation von Logistik- Materialfluss- und Produktionssystemen. Beuth Verlag, Berlin.

VDI, 2003. Richtlinie 2206: Entwicklungsmethodik für mechatronische Systemen. Beuth Verlag, Berlin.

Wang, G.G., 2002. Definition and review of virtual prototyping. Journal of Computing and Information Science in Engineering 2, 232–236. https://doi.org/10.1115/1.1526508

Weisser, R., 2008. Digitale Fabrik. ATZelektronik 3, 64–70. https://doi.org/10.1007/BF03223915

Wittek, S., Rausch, A., 2018. Learning state mappings in multi-level-simulation, in: Communications in Computer and Information Science. Springer Verlag, S. 208–218. https://doi.org/10.1007/978-3-319-96271-9_13

Wittek, S.H.A., Göttsche, M., Rausch, A., Grabowski, J., 2016. Towards multi-level-simulation using dynamic cloud environments, in: 2016 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH). IEEE, Lisbon, S. 1-7.

Wolfram, S., 1984. Universality and complexity in cellular automata. Physica D: Nonlinear Phenomena 10, 1–35. https://doi.org/10.1016/0167-2789(84)90245-8

Zackariasson, P., Wilson, T., 2012. The video game industry: Formation, present state, and future. Routedge, New York.

Zeigler, B.P., Muzy, A., Kofman, E. (Hrsg.), 2019. Theory of Modeling and Simulation, Third Edit. ed. Academic Press. https://doi.org/10.1016/C2016-0-03987-6

Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z., 2005. The Finite Element Method: Its Basis and Fundamentals Sixth edition.